PATENT

EMCR:069

EMC-00-175

APPLICATION FOR UNITED STATES LETTERS PATENT

for

Method of Pausing an MPEG Coded Video Stream

by

Sebastien Keller

Daniel Gardere

Jie Wang

Michel Noury

Sorin Faibish

EXPRESS MAIL MAILING LABEL NUMBER EL291391692US

BACKGROUND OF THE INVENTION

1. Field of the Invention

The present invention relates to processing and storage of compressed visual data, and in particular the processing and storage of compressed visual data for pausing and resuming transmission of an MPEG data stream.

2. Background Art

It has become common practice to compress audio/visual data in order to reduce the capacity and bandwidth requirements for storage and transmission. One of the most popular audio/video compression techniques is MPEG. MPEG is an acronym for the Moving Picture Experts Group, which was set up by the International Standards Organization (ISO) to work on compression. MPEG provides a number of different variations (MPEG-l, MPEG-2, etc.) to suit different bandwidth and quality constraints. MPEG-2, for example, is especially suited to the storage and transmission of broadcast quality television programs.

For the video data, MPEG provides a high degree of compression (up to 200:1) by encoding 8 x 8 blocks of pixels into a set of discrete cosine transform (DCT) coefficients, quantizing and encoding the coefficients, and using motion compensation techniques to encode most video frames as predictions from or between other frames. In particular, the encoded MPEG video stream is comprised of a series of groups of pictures (GOPs), and each GOP begins with an independently encoded (intra) I frame and may include one or more following P frames and B frames. Each I frame can be decoded without information from any preceding and/or following frame. Decoding of a P frame requires information from a preceding frame in the GOP. Decoding of a B frame requires information from both a preceding and a following frame in the GOP. To minimize decoder buffer requirements, transmission orders differ from presentation orders for some frames, so that all the information of the other frames required for decoding a B frame will arrive at the decoder before the B frame.

A GOP can be "open" or "closed." A GOP is closed if no prediction is allowed from any frame in a previous GOP. In other words, there are no B or P frames that require any information outside the GOP for decoding. A GOP is open if prediction is allowed from a frame in a previous GOP. In other words, there is a B or P frame that requires information in a frame outside of the GOP for decoding. In the typical case of an open GOP, the transmission order of the GOP begins with an I frame and has at least one B frame following the I frame. In the presentation order, this B frame precedes the first I frame in the GOP, and this B frame requires, for decoding, the last frame of a preceding GOP.

In addition to the motion compensation techniques for video compression, the MPEG standard provides a generic framework for combining one or more elementary streams of digital video and audio, as well as system data, into single or multiple program transport streams (TS) which are suitable for storage or transmission. The system data includes information about synchronization, random access, management of buffers to prevent overflow and underflow, and time stamps for video frames and audio packetized elementary stream packets embedded in video and audio elementary streams as well as program description, conditional access and network related information carried in other independent elementary streams. The standard specifies the organization of the elementary streams and the transport streams, and imposes constraints to enable synchronized decoding from the audio and video decoding buffers under various conditions.

The MPEG-2 standard is documented in ISO/IEC International Standard (IS) 13818-1, "Information Technology-Generic Coding of Moving Pictures and Associated Audio Information: Systems," ISO/IEC IS 13818-2, "Information Technology-Generic Coding of Moving Pictures and Associated Audio Information: Video," and ISO/IEC IS 13818-3, "Information Technology-Generic Coding of Moving Pictures and Associated Audio Information: Audio," which are incorporated herein by reference. A concise introduction to MPEG is given in "A Guide to MPEG Fundamentals and Protocol Analysis (Including DVB and ATSC)," Tektronix Inc., 1997, incorporated herein by reference.

One application of MPEG-2 coded video is video-on-demand (VOD). In a VOD application, a server streams MPEG-2 coded video in real time to a subscriber's decoder. The subscriber may operate a remote control providing well-known video cassette recorder (VCR) functions including play, stop, fast-forward, fast-reverse, and pause. In a typical implementation of the pause function, the server responds to a pause command from the subscriber by simply stopping transmission of the MPEG-2 coded video stream. Then the decoder loses synchronization, and the subscriber sees 0a result that is dependent on the performance of the decoder. The server responds to a resume command from the subscriber by resuming transmission of the MPEG-2 coded video stream. The decoder fails to present some of the video frames while re-synchronizing to the video stream. Another method that has been used is to pause by sending padding packets with continuous PCR stamps that will keep the decoder synchronized so that when the pause ends the decoder will not reset. This will create artifacts for a number of frames if the GOP is open.

SUMMARY OF THE INVENTION

The basic objective of the present invention is to provide a pause and resume function that delivers a valid MPEG data stream without video buffer underflow or overflow. Therefore, decoder synchronization is maintained and objectionable artifacts are avoided regardless of decoder performance.

In accordance with one aspect, the invention provides a method of pausing an MPEG coded video stream. The MPEG coded video stream includes a series of groups of pictures. Each group of pictures (GOP) includes an I frame and a plurality of B or P frames. The method includes selecting an I frame from the MPEG coded video stream, and constructing a pause GOP from the selected I frame. The pause GOP includes an I frame, freeze frames, and padding. The method further includes making a seamless transition from the MPEG coded video stream to the pause GOP, and playing the pause GOP a plurality of times in succession.

In accordance with another aspect, the invention provides a method of pausing an MPEG-2 coded video stream including a series of groups of pictures. Each group of pictures (GOP) includes an I frame and a plurality of B or P frames. The method includes selecting an I frame from the MPEG-2 coded video stream, and constructing a pause GOP from the selected I frame. The pause GOP includes an I frame and a number of dual-motion frozen P frames and padding to obtain a desired frame rate when the pause GOP is played a plurality of times in succession. The dual-motion frozen P frames presents a top field and a bottom field that is substantially the same as the top field. The method further includes making a seamless transition from the MPEG-2 coded video stream to the pause GOP, and playing the pause GOP a plurality of times in succession, while inserting into the MPEG-2 stream a selected amount of padding to obtain a desired constant bit rate, and restamping PTS, DTS, and continuity counter values in the MPEG-2 stream.

BRIEF DESCRIPTION OF THE DRAWINGS

Other objects and advantages of the invention will become apparent upon reading the following detailed description with reference to the accompanying drawings, in which:

FIG. 1 is a block diagram of a data network including a video file server implementing various aspects of the present invention;

FIG. 2 is a flowchart of a procedure executed by a stream server computer in the video file server of FIG. 1 to service client requests;

FIG. 3 is a flowchart of a procedure for splicing MPEG clips;

FIG. 4 is a flowchart of a procedure for seamless video splicing of MPEG clips;

FIG. 5 is a more detailed flowchart of the procedure for seamless video splicing of MPEG clips;

FIG. 6 is a continuation of the flowchart begun in FIG. 5;

FIG. 7 is a timing diagram showing a timing relationship between video presentation units (VPUs) and associated audio presentation units (APUs) in an original MPEG-2 coded data stream;

FIG. 8 is a timing diagram showing a timing relationship between video presentation units (VPUs) and associated audio presentation units (APUs) for a fast-forward trick-mode stream;

FIG. 9 is a flowchart of a procedure for selection and alignment of audio presentation units (APUs) in the fast-forward trick-mode stream;

FIG. 10 is a flowchart of a procedure for producing a trick-mode MPEG-2 transport stream from a regular MPEG-2 transport stream (TS);

FIG. 11 is a diagram illustrating relationships between the MPEG discrete cosine transform (DCT) coefficients, spatial frequency, and the typical zig-zag scan order;

FIG. 12 is a diagram illustrating a relationship between an MPEG-2 coded bit stream and a reduced-quality MPEG-2 coded bit stream resulting from truncation of high-order DCT coefficients;

FIG. 13 is a flowchart of a procedure for scaling MPEG-2 coded video using a variety of techniques;

FIG. 14 is a flowchart of a procedure for signal-to-noise ratio scaling MPEG-2 coded video using a frequency-domain low-pass truncation (FDSNR_LP) technique;

FIG. 15 is a flowchart of a procedure for signal-to-noise ratio scaling MPEG-2 coded video using a frequency-domain largest-magnitude coefficient selection (FDSNR_LM) technique;

FIG. 16 is a flowchart of a procedure that selects one of a number of techniques for finding a certain number "k" of largest values out of a set of "n" values;

FIG. 17 is a flowchart of a procedure for finding a certain number “k” of largest values from a set of “n” values, which is used in the procedure of FIG. 16 for the case of k<<½ n;

FIG. 18 is a diagram of a hash table and associated hash lists;

FIG. 19 is a flowchart of a procedure for finding a certain number “k” of values that are not less than the smallest of the “k” largest values in a set of “n” values beyond a certain amount.

FIG. 20 is a flowchart of modification of the procedure of FIG. 15 in order to possibly eliminate escape sequences in the (run, level) coding of the largest magnitude coefficients;

FIG. 21 is a flowchart of a subroutine called in the flowchart of FIG. 20 in order to possibly eliminate an escape sequence;

FIG. 22 is a first portion of a flowchart of a procedure for scaling an MPEG-2 coded video data stream using the modified procedure of FIG. 20 while adjusting the parameter “k” to achieve a desired bit rate, and adjusting a quantization scaling factor (QSF) to achieve a desired frequency of occurrence of escape sequences;

FIG. 23 is a second portion of the flowchart begun in FIG. 22;

FIG. 24 is a simplified block diagram of a volume containing a main file, a corresponding fast forward file for trick mode operation, and a corresponding fast reverse file for trick mode operation;

FIG. 25 is a more detailed block diagram of the volume introduced in FIG. 24;

FIG. 26A is a diagram showing video file access during a sequence of video operations including transitions between the main file, the related fast forward file, and the related fast reverse file;

FIG. 26B shows a script of a video command sequence producing the sequence of video play shown in FIG. 26A;

FIG. 27 is a table of read and write access operations upon the volume of FIG. 24 and access modes that are used for the read and write access operations;

FIG. 28 is a hierarchy of video service classes associated with the fast forward file and the fast reverse file in the volume of FIG. 25;

FIG. 29 is a block diagram showing a way of programming and configuring a video file server for producing a seamless MPEG-2 coded video stream including pauses;

FIG. 30 is a block diagram showing content of an elementary stream including original video;

FIG. 31 is a block diagram showing content of an elementary stream and a transport stream for a P-freeze frame;

FIG. 32 is a block diagram showing content of a pause GOP in an elementary stream, in a packetized elementary stream, and in a transport stream before and after a transport stream multiplexor in FIG. 29;

FIG. 33 is a first sheet of a flowchart showing how an active pause is performed on an I-frame in a stream of MPEG-2 coded video;

FIG. 34 is a second sheet of the flowchart begun in FIG. 33;

FIG. 35 is a block diagram showing a preferred construction for normal play video buffers and pause video buffers introduced in FIG. 29;

FIG. 36 shows a graph of the level of a video buffer verifier as a function of time during an MPEG-2 coded video stream including normal play, a first pause, a seek, a second pause, and a resumption of the normal play;

FIG. 37 shows a sequence of video frames in an original stream of MPEG-2 coded data;

FIG. 38 shows how the original stream of FIG. 37 is modified during an active pause upon a closed GOP for a first case of a play followed by a pause;

FIG. 39 shows how the original stream of FIG. 37 is modified for an active pause upon a closed GOP for a second case of a play followed by a pause, followed by a seek;

FIG. 40 shows how the original stream of FIG. 37 is modified for an active pause upon an open GOP for a first case of a play followed by a pause;

FIG. 41 shows how the original stream of FIG. 37 is modified for an active pause upon an open GOP for a second case of a play followed by a pause, followed by a seek;

FIG. 42 is a block diagram showing various program objects for implementing the configuration of FIG. 29 in the video file server of FIG. 1;

FIG. 43 shows a preferred way of implementing a P-freeze frame in order to freeze a single field of an I-frame;

FIG. 44 shows transcoding of an I-frame of a pause GOP for eliminating flicker;

FIG. 45 shows how the transcoding of FIG. 44 is performed for either picture coded frames or field coded frames, and for field DCT or for frame DCT for picture coded frames;

FIG. 46 further shows use of field line replacement for the case of a pause GOP including an I frame that is picture coded with frame DCT;

FIG. 47 shows how the field line replacement of FIG. 46 could be performed by decoding and re-encoding the pixels of the I frame;

FIG. 48 shows how the field line replacement is performed in the DCT domain by a linear transformation such as a matrix multiplication;

FIG. 49 is a flow diagram for the field line replacement in the DCT domain using the matrix multiplication of FIG. 48;

FIG. 50 shows a sequence of frames produced in a two-step transcoding method for reducing flicker during a pause;

FIG. 51 is a flowchart for progressive transcoding and two-step transcoding for reducing flicker during a pause; and

FIG. 52 shows a preferred alignment of audio presentation units with video presentation units during a pause.

While the invention is susceptible to various modifications and alternative forms, specific embodiments thereof have been shown by way of example in the drawings and will be described in detail. It should be understood, however, that it is not intended to limit the form of the invention to the particular forms shown, but on the contrary, the intention is to cover all modifications, equivalents, and alternatives falling within the scope of the invention as defined by the appended claims.

DESCRIPTION OF ILLUSTRATIVE EMBODIMENTS

With reference to FIG. 1, there is shown a block diagram of a data network 20 linking a number of clients 21, 22, 23 to a video file server 24 implementing various aspects of the present invention. The video file server 24 includes at least one stream server computer 25 and a data storage system 26. The stream server computer 25 has a processor 27 and a network link adapter 28 interfacing the processor to the data network 20. The processor 27 executes a data streaming program 29 in memory 30 in order to stream MPEG coded video in real-time to the clients.

Client requests for real-time video are placed in client play lists 31 in order to schedule in advance video file server resources for the real-time streaming of the MPEG coded video. The play lists 31 specify a sequence of video clips, which are segments of MPEG-2 files 32, 33 in data storage 34 of the data storage system 26. The stream server processor 27 accesses a client play list in advance of the time to begin streaming MPEG coded video from a clip, and sends a video prefetch command to a storage controller 35 in the data storage system 26. The storage controller responds to the video prefetch command by accessing the clip in the data storage 34 to transfer a segment of the clip to cache memory 36. When the video data of the segment needs to be sent to the client, the stream server processor 27 requests the data from the storage controller 35, and the storage controller immediately provides the video data from the cache memory 36. Further details regarding a preferred construction and programming of the video file server 24 are disclosed in Duso et al., U.S. Patent 5,892,915 issued Apr. 6, 1999, entitled "System Having Client Sending Edit Commands to Server During Transmission Of Continuous Media From One Clip in Play List for Editing the Play List," incorporated herein by reference.

In accordance with an aspect of the invention, the stream server computer 25 executes an MPEG scaling program 38 to produce reduced-quality MPEG coded video from nonscalable MPEG-2 coded video by truncating discrete cosine transform (DCT)

AC coefficients from the coded blocks in the MPEG-2 coded video data. The reduced-quality MPEG coded video can be produced during ingestion of an MPEG-2 file 32 from the network 20, and stored in one or more associated files 37. Alternatively, the reduced-quality MPEG coded video in the files 37 could be produced as a background task from the MPEG-2 file 32. Reduced-quality MPEG coded video could also be produced in real-time from an MPEG-2 file 33 during streaming of the reduced-quality MPEG coded video from the stream server computer 25 to the network 20. The reduced-quality MPEG coded video is useful for a variety of applications, such as browsing and review of stored MPEG-2 assets for search and play-list generation, bit stream scaling for splicing, and bit-rate adjustment via video quality alteration for services with limited resources.

A typical example of browsing for play-list generation involves searching stored assets in a multi-media data base for segments of a desired content to be included in the play list, and in particular selecting the beginning frame and ending frame of each segment to be included. Such editing occurs often in the broadcast environment for inserting commercials and news clips into pre-recorded television programming, and for editing movies for content and time compression. The decoding technique of the present invention permits a PC workstation 23 to perform the decoding and display in real-time by execution of a software program. An operator can view the video content in a display window 39 in a fast-forward or fast-reverse mode, stop at and resume from freeze frames that are valid "in points" and "out points" for seamless splicing, and select an in-point and out-point for a next segment to be included in the play list. The stream server computer 25 could also include a seamless splicing program 40 providing seamless transitions between video segments that are contiguous in a play list and are from different video clips.

For seamless splicing, it is often necessary to reduce the bitrate for one or more frames at the end of a first segment prior to splicing to a second segment. In this case the bitrate must be reduced to avoid buffer overflow as a result of displaying the original frames at the end of the first segment. One method of reducing the bitrate is to insert a freeze frame at the end of the first segment, but this has the disadvantage of introducing distortion in the temporal presentation of the frames and precluding frame accuracy. A less disruptive method is to use the present invention for reducing the bitrate for a lower-quality presentation of one or more frames at the end of the first segment.

The present invention can also reduce the bit transmission rate and storage requirements for MPEG-2 applications by altering the video quality. For example, different clients may present different bandwidth access requests for video from nonscalable MPEG-2 files 32, 33 in the video file server. Also, temporary network congestion may limit the bandwidth available to satisfy a request for real-time streaming of video data. In each case, the present invention can alter the video quality to meet the desired or available bandwidth to satisfy the request.

With reference to FIG. 2, there is shown a flowchart of a procedure executed by a stream server computer in the video file server of FIG. 1 to service client requests. In a first step 50, execution branches to step 51 when a client request is not a request for real-time streaming. If the request is a request to input a new MPEG-2 file, then execution branches to step 52 to input the new MPEG-2 file and to create a reduced-quality version of the MPEG-2 file as available resources permit. If the request is not a request to input a new MPEG-2 file, then execution continues from step 51 to step 53. In step 53, execution branches to step 54 if the request is for play list editing. In step 54, the client may browse through the reduced-quality MPEG file to select in-points and out-points of clips to be spliced.

In step 50, when the request is for real-time streaming, then execution branches to step 55. In step 55, if there is network congestion so that there is insufficient bandwidth to transmit a stream of original-quality MPEG-2 coded video, then execution branches to step 56 to stream compressed video from the reduced-quality MPEG file. If no reduced-quality MPEG file is available for the desired clip, then the reduced-quality MPEG coded video to be streamed is produced in real-time from the original-quality MPEG-2 coded video. There are also applications, such as the display of spatially down-sampled video in a small display window (39 in FIG. 1), for which the client may request reduced-quality MPEG coded video. In this case, in the absence of network congestion, execution will continue from step 55 to step 57, and branch from step 57 to step 56 for streaming of reduced-quality MPEG coded video to the client.

Reduced-quality MPEG coded video is also useful for "trick-mode" operation. Trick-mode refers to fast forward or fast reverse display of video, in a fashion analogous to the fast forward and fast reverse playback functions of a video cassette recorder (VCR). The problem with trick-mode operation is that the speed of the MPEG stream cannot simply be speeded up because the transmission bandwidth would be excessive and a conventional MPEG-2 decoder will not be able to handle the increased data rate or even if the decoder would have been able to support the increased data rate, such a change in the original operating conditions is not allowable. For this reason, in trick-mode, neither the original display rate of 29.97 frames per second (for NTSC or 25 frames per second for PAL) nor the original transport stream (TS) multiplex rate should change. Nor is it possible to simply decimate frames since only the I frames are independently coded, and the P frames and B frames need the content of certain other frames for proper decoding. The I frames typically occur once for every 15 frames. Assuming that this convention is followed in the encoding process, it would be possible to preserve and play each I frame from each and every group of pictures (GOP), resulting in a 15 times slower temporal sampling rate, or a 1 to 15 speeding up of motion if the I frames only are played back at the nominal NTSC rate of approximately 30 frames per second. Consequently, the content of a 60 minutes duration clip will be covered in 4 minutes. Unfortunately the average information content per frame for the I frames is more than four times the average information content of the P and B frames. Therefore, the trick-mode cannot be implemented simply by transmitting only the I frames for a speed-up by a factor of 15, because this would need an increase in the TS multiplex rate over the nominal rate.

In particular, the average information content of an I frame has been measured to be about 56,374.6 bytes. If the I frames only are transmitted at the standard NTSC rate, then the bit transmission rate would be: 8(bits per byte) * 56,374.6(bytes per frame) * 29.97(frames per sec.) or about 13,516,374.1 bits per second only for the video stream, which is significantly above - almost 3.38 times - the original rate of 4 megabits per second used in this test. This calculation, being based on an average quantity, is ignoring the indispensable need for an actually higher transport rate to provide some safety margin to handle short-term-sustained large size I frame chains (bursts) which practically always happen. Clearly, some form of modification in the trick-mode operation definition is required to handle this problem and pull the bit-rate requirement down to the nominal 4 megabits per second.

Two degrees of freedom are available to achieve such a reduction in the required bit-rate for trick-mode operation. The first is I frame compression quality and the second is a motion speed-up ratio. With respect to compression quality, it is well known that human observers' perception of image detail degrades with increasing motion speed of objects in the scene. Based on this fact, the type of D pictures were introduced in MPEG-1 video syntax for fast visible (forward or reverse) search purposes. (See ISO/IEC 11172-2: 1993 Information Technology - Coding of moving pictures and associated audio for digital storage media at up to about 1.5 Mbits/s - Part 2: Video, Annex D.6.6. Coding D-Pictures, p. 102). D pictures make use of only the DC coefficients in intra coding to produce very low quality (in terms of SNR) reproductions of desired frames which were judged to be of adequate quality in fast search mode.

In order to provide support for enhanced quality trick-mode operation, the quality of the original I frames can be reduced by the preservation of just a sufficient number of AC DCT coefficients to meet the bit-rate limitation. Based on experiments with two standard video test sequences (one encoded at 15 Mbits/sec. and the other at 24 Mbits/sec. and both with I frames only), it is observed that the bandwidth for I frames can be scaled to one half by keeping about 9 lowest order AC coefficients and eliminating the rest. This scheme provides good quality even at the full spatial and temporal resolution, much better than D pictures.

The inherent speed-up ratio lower bound imposed by the GOP structure can be relaxed and further lowered by freeze (P) frame substitution in between genuine (SNR scaled or non-scaled) I frames. The maximum number of freeze frames that can be inserted before visually disturbing motion jerkiness occurs, is very likely to depend heavily on the original GOP structure (equivalently the separation between I frames of the original sequence) and the original amount of motion in the clip. However, 1, 2 or 3 freeze frame substitutions in between genuine I frames present reasonable choices which will yield speed-up ratios of 1 to 7.5, 1 to 5 and 1 to 3.75 respectively instead of the 1 to 15 speed-up ratio provided by the genuine I frames only implementation. (These ratios are computed by a first-order approximation that neglects a slight increase in bandwidth required by the consecutive freeze frames, which are inserted in between genuine I frames and can typically be made very small in size in comparison to the average size of a genuine I frame. Therefore, the insertion of 1, 2, 3 freeze frames will result in bandwidth reductions of 2 to 1, 3 to 1 and 4 to 1 respectively. The accuracy of this approximation degrades as more consecutive freeze frames and/or SNR scaling is employed.) An easy way to see the validity of these approximate figures is to note for example that in the case of 1 freeze frame insertion, the total presentation time of the trick-mode clip for an originally 60 minutes duration asset will increase from 4 minutes to 8 minutes. Since due to the underlying assumption of the first-order approximation stated above, the same amount of data (I frames only) will be transmitted in this doubled time interval, the bandwidth requirement will be halved. The final choice for trick-mode implementation should reflect a balanced trade-off along these two degrees of freedom. For example, SNR scaling of I frames down to 9 AC coefficients can be used along with single freeze frame insertion between I frames. These two choices, both of which are individually capable of providing a 2 to 1 bandwidth reduction as discussed before, will yield a combined 4 to 1 bandwidth reduction which will comfortably bring the non-scaled I frame-only bit-rate of 13,516,374.1 bits/sec. down to below the 4 Mbits/sec. quota. If the visual quality provided by 9 AC coefficients is not considered adequate, then SNR scaling could be tuned to keep more AC coefficients at the expense of a smaller bandwidth reduction. This, however, could be compensated consequently by increasing the number of freeze frames to be used in between I frames. Coarser quantization (and therefore poorer visual quality) can be tolerated at high trick-mode speeds and better visual quality should be retained at lower trick-mode speeds.

With reference to FIG. 2, if the client has requested trick-mode operation, execution branches from step 58 to step 59. In step 59, execution branches to step 60 for a low value of speed-up. In step 60, the trick-mode stream is produced by streaming original-quality I frames and inserting three freeze frames per I frame, to yield a speed-up factor of 15/4= 3.75 based on an original MPEG-2 coded stream having one I frame for every 15 frames. For a higher speed-up factor, execution branches from step 59 to step 61. In step 61, either one or two freeze frames are selected per I frame to provide a speed-up factor of 15/2=7.5, or 15/3 = 5 respectively. Then in step 62 the trick-mode stream is produced by streaming reduced-quality I frames and inserting the selected number of freeze frames between the reduced-quality I frames. If a trick-mode operation is not requested in step 58, then execution continues from step 58 to step 63. In step 63, the stream server computer streams original-quality MPEG-2 coded data to the client. Further details regarding trick-mode operation are described below with reference to FIGs. 7 to 10.

FIGs. 3 to 6 show further details regarding use of the present invention for MPEG splicing. In particular, reduced-quality frames are substituted for the freeze frames used in the seamless splicing procedure found in the common disclosure of Peter Bixby et al., U.S. application Ser. 09/539,747 filed March 31, 2000; Daniel Gardere et al., U.S. application Ser. 09/540,347 filed March 31, 2000; and John Forecast et al., U.S. application Ser. 09/540,306 filed March 31, 2000; which are all incorporated by reference herein. The common disclosure in these U.S. applications considered pertinent to the present invention is included in the written description below with reference to FIGs. 3 to 6 in the present application (which correspond to FIGs. 19, 22, 23, and 24 in each of the cited U.S. applications).

FIG. 3 shows a basic procedure for MPEG splicing. In the first step 121, the splicing procedure receives an indication of a desired end frame of the first clip and a desired start frame of the second clip. Next, in step 122, the splicing procedure finds the closest I frame preceding the desired start frame to be the In Point for splicing. In step 123, the splicing procedure adjusts content of the first clip near the end frame of the first clip and adjusts content of the second clip near the In Point in order to reduce presentation discontinuity (due to decoder buffer underflow) and also to prevent decoder buffer overflow when decoding the spliced MPEG stream. Finally, in step 124, the concatenation of the first clip up to about the Out Point and the second clip subsequent to about the In Point is re-formatted, including re-stamping of the presentation time stamps (PTS), decoding time stamps (DTS), and program clock reference (PCR) values for the audio and video streams in the second clip.

Considering now video splicing, the splicing procedure should ensure the absence of objectionable video artifacts, preserve the duration of the spliced stream, and if possible, keep all of the desired frames in the spliced stream. The duration of the spliced stream should be preserved in order to prevent any time drift in the scheduled play-list. In some cases, it is not possible to keep all of the original video frames due to buffer problems.

Management of the video buffer is an important consideration in ensuring the absence of objectionable video artifacts. In a constant bit rate (CBR) and uniform picture quality sequence, subsequent pictures typically have coded representations of drastically different sizes. The encoder must manage the decoder's buffer within several constraints. The buffer should be assumed to have a certain size defined in the MPEG-2 standard. The decoder buffer should neither overflow nor underflow. Furthermore, the decoder cannot decode a picture before it receives it in full (i.e. completely). Moreover, the decoder should not be made to "wait" for the next picture to decode; this means that every 40 ms in PAL and 1/29.97 second in NTSC, the decoder must have access to a full picture ready to be decoded.

The MPEG encoder manages the video decoder buffer through decode time stamps (DTS), presentation time stamps (PTS), and program clock reference (PCR) values. When splicing the end of a first clip to the beginning of a second clip, there will be a problem of video buffer management if a duration of time DTS_{L1}-T_e is different from a duration of time DTS_{F2}-PCR_{e2} minus one video frame (presentation) interval, where DTS_{L1} is the DTS at the end of the first clip and indicates the time at which the video decoder buffer is emptied of video data from the first clip, T_e is the time at which the last video frame's data is finished being loaded into the video decoder buffer, DTS_{F2} is the DTS of the first frame of the second clip, and PCR_{e2} is the PCR of the second clip extrapolated from the value of the most recent received genuine PCR record, to the first byte of the picture header sync word of the first video frame in the clip to start. The extrapolation adjusts this most recently received genuine PCR record value by the quotient of the displacement in data bits of the clip from the position where it appears in the second clip to the position at which video data of the first frame of the second clip begins, divided by the data transmission bit rate for transmission of the clip to the decoder. Because the time PCR_{e2} must immediately follow T_e, there will be a gap in the decoding and presentation of video frames if DTS_{F2}-PCR_{e2} is substantially greater than DTS_{L1}-T_e plus one video frame interval. In this case, the buffer will not be properly full to begin decoding of the second clip one video frame interval after the last frame of the first clip has been decoded. Consequently, either the second clip will be prematurely started to be decoded or the decoder will be forced to repeat a frame one or more times after the end of the display of the last frame from the first clip to provide the required delay for the second clip's buffer build-up. In the case of a premature start for decoding the second clip, a video buffer underflow risk is generated. On the other hand, in case of repeated frames, the desired frame accuracy for scheduled play-lists is lost besides the fact that neither a precise timing adjustment can be achieved through this procedure.

If DTS_{F2}-PCR_{e2} is substantially less than DTS_{L1}-T_e plus one video frame interval, then the decoder will not be able to decode the first frame of the second clip at the specified time DTS_{F2} because the last frame of the first clip will not yet have been removed from the video buffer. In this case a video buffer overflow risk is generated.

Video buffer overflow may present a problem not only at the beginning of the second clip, but also at a subsequent location of the second clip. If the second clip is encoded by an MPEG-2 compliant encoder, then video buffer underflow or buffer overflow will not occur at any time during the decoding of the clip. However, this guarantee is no longer valid if the DTS_{F2}-PCR_{e2} relationship at the beginning of the second clip is altered. Consequently, to avoid buffer problems, the buffer occupancy at the end of the first clip must be modified in some fashion. This problem is inevitable when splicing between clips having significantly different ending and starting buffer levels. This is why the Society of Motion Picture and Television Engineers (SMPTE) has defined some splice types corresponding to well-defined buffer levels. (See SMPTE Standard 312M, entitled "Splice Points for MPEG-2 Transport Streams," SMPTE Journal, Nov. 1998.) In order to seamlessly splice the first clip to the second clip, the content of the first clip (towards its end) is modified so that PCR_{e2} can immediately follow T_e (by one byte transmission time) and DTS_{F2} can just follow DTS_{L1} (by one video frame presentation interval).

FIG. 4 shows a flow chart of a seamless video splicing procedure that attains the desired condition just described above. In a first step 141, the first DTS of the second clip is anchored at one frame interval later than the last DTS of the first clip in order to prevent a video decoding discontinuity. Then, in step 142, the procedure branches depending on whether the PCR extrapolated to the beginning frame of the second clip falls just after the ending time of the first clip. If so, then the splice will be seamless with respect to the original video content. Otherwise, the procedure branches to step 143. In step 143, the content of the first clip is adjusted so that the PCR extrapolated to the beginning frame of the second clip falls just after the ending time of the first clip. Therefore the desired conditions for seamless video splicing are achieved.

With reference to FIG. 5, there is shown a more detailed flow chart of a seamless video splicing procedure. In a first step 151, the procedure inspects the content of the first clip to determine the last DTS/PTS of the first clip. This last DTS/PTS of the first clip is designated DTS_{L1}. Next, in step 152, the procedure inspects the content of the first clip to determine the time of arrival (T_e) of the last byte of the first clip. In step 153, the procedure adds one frame interval to DTS_{L1} to find the desired first DTS location for the second clip. The sum, designated DTS_{F1}, is equal to DTS_{L1} +1/FR, where FR is the video frame rate. In step 154, while keeping the DTS-PCR_e relationship unaltered for the second clip, the procedure finds the time instant, designated T_S, at which the first byte of the second clip should arrive at the decoder buffer. This is done by calculating $T_{START}=DTS_{F2}-PCR_{e2}$, and $T_S=DTS_{F1}-T_{START}$.

Continuing in FIG. 6, in step 155, execution branches depending on whether T_S is equal to T_e plus 8 divided by the bit rate. If not, then the clips to be spliced need modification before concatenation, and execution branches to step 156. In step 156, execution branches depending on whether T_S is less than T_e plus 8 divided by the bit rate. If not, then there is an undesired gap in between the clips to be spliced, and execution branches to step 157. In step 157, null packets are inserted into the clips to be spliced to compensate for the gap. The gap to be compensated has a number of bytes, designated G_r, equal to (T_S-T_e)(BIT RATE)/8 minus one. If in step 156, T_S is less than T_e plus 8 divided by the bit rate, then execution continues from step 156 to step 158 to open up a certain amount of space in the first clip to achieve $T_S=T_e+8/(\text{BIT RATE})$. The number of bytes to drop is one plus $(T_e\text{-}T_S)$(BIT RATE)/8. If possible, the bytes are dropped by removing null packets. Otherwise, one or more frames at the end of the first clip are replaced with corresponding reduced-quality frames, which have fewer bytes than the original-quality frames at the end of the first clip.

If in step 155 T_S is found to be equal to T_e plus 8 divided by the bit rate, then execution continues to step 159. Execution also continues to step 159 from steps 157 and 158. In step 159, the transport streams from the two clips are concatenated. Finally, in step 160, a subroutine is called to compute a video time stamp offset, designated as V_{OFFSET}. This subroutine finds the DTS of the last video frame (in decode order) of the first clip. This DTS of the last video frame of the first clip is denoted DTS_{VL1}. Then the subroutine finds the original DTS of the first frame to be decoded in the second clip. This DTS of the first frame to be decoded in the second clip is denoted DTS_{VF2}. Finally, the subroutine computes the video time stamp offset V_{OFFSET} as $DTS_{VL1}\text{-}DTS_{VF2}$ plus one video frame duration.

FIGs. 7 to 10 show further details regarding trick-mode operation. FIG. 7 shows a timing relationship between video presentation units (VPUs) and associated audio presentation units (APUs) in an original MPEG-2 coded data stream, and FIG. 8 shows similar timing for the fast-forward trick-mode stream produced from the original data stream of FIG. 7. (The fast-forward trick-mode stream is an example of a trick-mode stream that could be produced in step 60 of FIG. 2.) The original data stream has successive video presentation units for video frames of type I, B, B, P, B respectively. The trick-mode stream has successive video presentation units for video frames of types I, F, F, I, F where "F" denotes a freeze P (or possibly B) frame. Each I frame and immediately following F frames produce the same video presentation units as a respective I frame in the original data stream of FIG. 7, and in this example, one in every 15 frames in the original data stream is an I frame. Each freeze frame is coded, for example, as a P frame repeating the previous I frame or the previous P-type freeze-frame (in display order). In each freeze frame, the frame is coded as a series of maximum-size slices of macroblocks, with an initial command in each slice indicating that the first macroblock is an exact copy of the corresponding macroblock in the previous frame (achieved by predictive encoding with a zero valued forward motion compensation vector and no encoded prediction error), and two consequent commands indicating that the following macroblocks in the slice until and including the last macroblock of the slice are all coded in the same way as the first macroblock.

For trick-mode operation, there is also a problem of how to select audio presentation units (APU) to accompany the video presentation units that are preserved in the trick-mode stream. Because the video presentation units (VPU) have a duration of (1/29.97) sec. or about 33.37 msec. and the audio presentation units (APU) have a duration of 24 msec., there is neither a one-to-one correspondence nor alignment between VPUs and APUs. In a preferred implementation, the audio content of a trick-mode clip is constructed as follows. Given the total presentation duration (1/29.97) sec. or about 33.37 msec. for a single video frame, it is clear that always at least one and at most two 24 msec. long audio presentation units (APU) will start being presented during the end-to-end presentation interval of each video frame. This statement refers to the original clip and does not consider any audio presentation unit whose presentation is possibly continuing as the video frame under consideration is just put on the display. The first of the above mentioned possibly two audio presentation units will be referred to as the aligned audio presentation unit with respect to the video frame under consideration. For example, in FIG. 8, the APU_j is the aligned audio presentation unit with respect to the VPU_i. Now, when the I frames are extracted and possibly SNR scaled and possibly further interleaved with a number of freeze P frames in between them to produce the trick-mode video packetized elementary stream (PES), the associated trick-mode audio stream is constructed as follows. For each I type video frame presentation interval (and for that matter also for freeze P type video frames) in this trick-mode clip, the above stated fact of at least one (and at most two) audio presentation unit being started, holds. Then for each I frame presentation interval in the trick-mode clip, once any possibly previously started and continuing audio presentation unit ends, insert its aligned audio presentation unit (from the original clip) and continue inserting APUs from the original clip subsequent to the aligned one until covering the rest of the I frame presentation interval and also any possibly following freeze P frame presentation intervals until crossing into and overlapping (or less likely aligning) with the next I frame's presentation interval. In FIG. 8, for example, the audio presentation units APU_j, APU_{j+1}, APU_{j+2}, and APU_{j+3} are inserted, until crossing into and overlapping with the next I frame VPU_{i+15}. Following APU_{j+3} is inserted APU_k, which designates the APU aligned with VPU_{i+15} in the original stream. Clearly, the final alignment of (the aligned and consequent) audio presentation units with respect to their associated I frames will be slightly different in the trick-mode clip as compared to the original clip. However, considering how the trick-mode audio component will sound like, this poses no problem at all.

FIG. 9 is a flowchart of a procedure for producing the desired sequencing of audio presentation units (APUs) in the fast-forward trick-mode stream. This procedure scans the audio elementary stream in the original MPEG-2 stream to determine the sequence of APUs in the original stream and their presentation-time alignment with the I frames in the video elementary stream of the original MPEG-2 transport stream, while selecting APUs to include in the trick-mode stream. In a first step 171, execution proceeds once the end of the current APU is reached. If the end of the current APU has not entered a new VPU (*i.e.*, the beginning of the current APU is within the presentation time of one VPU and the end of the current APU is within the presentation time of the same VPU), or if it has entered a new VPU (*i.e.*, the beginning of the current APU is within the presentation time of one VPU and the end of the current APU is within the presentation time of a new (next) VPU) but the new VPU is not an I frame, then execution branches to step 174. In step 174, an APU pointer is incremented, and in step 175 execution proceeds into this next APU. If in step 173 the end of the current APU extends into an I frame, then in step 176 the APU pointer is advanced to point to the first APU beginning within the duration of the VPU of the I frame in the original MPEG-2 stream.

FIG. 10 is a flowchart of a procedure for producing a trick-mode stream from an MPEG-2 transport stream (TS). In a first step 181, the MPEG-2 TS is inputted. In step 182, the video elementary stream (VES) is extracted from the TS. In step 183, a concurrent task extracts the audio elementary stream (AES) from the TS. In step 184, I frames are extracted from the VES and valid packetized elementary stream (PES) packets are formed encapsulating the I frames. In step 185, the I frames are SNR scaled, for the high speed cases of the trick-mode. In step 186, P-type freeze frames are inserted into the stream of SNR scaled I frames (in between the scaled I frames), and valid PES packets are formed for the trick-mode VES encapsulating the P-type freeze frames and the SNR scaled I frames. Concurrently, in step 187, appropriate audio access units (from the originally input MPEG-2 TS asset) are selected and concatenated based on the structure of the VES being formed for the trick-mode clip, as described above with reference to FIG. 9, and valid PES packet encapsulation is formed around these audio access units. Finally, in step 188, the trick-mode TS stream is generated by multiplexing the trick-mode VES from step 186 into a system information (SI) and audio PES carrying TS skeleton including the audio PES packets from step 187.

FIGs 11 to 19 include details of the preferred techniques for truncating AC DCT coefficients for producing low-quality MPEG coded video from original-quality MPEG-2 coded video. Most of these techniques exploit the fact that in the typical (default) zig-zag scan order, the basis functions for the high-order AC DCT coefficients have an increasing frequency content. FIG 11, for example, shows a matrix of the DCT coefficients C_{ij}. The row index (i) increases with increasing vertical spatial frequency in a corresponding 8x8 coefficient block, and the column index (j) increases with increasing horizontal spatial frequency in the corresponding 8x8 coefficient block. The coefficient C_{11} has zero frequency associated with it in both vertical and horizontal directions, and therefore it is referred to as the DC coefficient of the block. The other coefficients have non-zero spatial frequencies associated with their respective basis functions, and therefore they are referred to as AC coefficients. Each coefficient has an associated basis function $f_{ij}(x,y)$ that is separable into x and y components such that $f_{ij}(x,y)= f_i(y)f_j(x)$. The x and y component functions $f_i(y)$ and $f_j(x)$ are shown graphically in FIG. 11 as cosine functions in order to illustrate their associated spatial frequencies. In practice, the component functions are evaluated at discrete points for the 64 pixel positions in the 8x8 blocks, so that each of the DCT basis functions is an 8x8 array of real numbers. In particular, the component functions are:

$$f_i(y) = \frac{1}{2}C(i-1)\cos\left(\frac{(2y-1)(i-1)\pi}{16}\right) \text{ for y=1, 2, 3, …, 8}$$

$$f_j(x) = \frac{1}{2}C(j-1)\cos\left(\frac{(2x-1)(j-1)\pi}{16}\right) \text{for x=1, 2, 3, …, 8}$$

With $$C(i) = \begin{cases} \frac{1}{\sqrt{2}}, & if \quad i = 0 \\ 1, & otherwise \end{cases}$$

The heavy black line through the matrix of coefficients in FIG. 11 denotes the default zig-zag scan order typically used for MPEG-2 encoding. Listed in this order, the coefficients are C_{11}, C_{12}, C_{21}, C_{31}, C_{22}, C_{13}, C_{14}, C_{23}, C_{32}, C_{41}, …, C_{86}, C_{77}, C_{68}, C_{78}, C_{87}, C_{88}. The first coefficient in this zig-zag scan order is the DC coefficient C_{11} providing the lowest spatial frequency content in the 8x8 block of pixels, and the last coefficient in this zig-zag scan order is the coefficient C_{88} providing the highest spatial frequency content in the 8x8 block of pixels.

FIG. 12 is a diagram illustrating a relationship between an original MPEG-2 coded bit stream 200 and a reduced-quality MPEG-2 coded bit stream 210 resulting from truncation of high-order DCT coefficients from the original MPEG-2 coded bit stream. Shown in the original MPEG-2 coded bit stream 200 is a portion of a video PES packet including DCT coefficients for an 8x8 pixel block. The DCT coefficients include a differentially coded DC coefficient 201, and three (run, level) events 202, 203, 204 encoding three respective nonzero AC coefficients possibly along with some zero valued AC coefficients preceding the three nonzero valued ones. The DCT coefficients are ordered according to the zig-zag scan order shown in FIG. 11 (or possibly according to an alternate zig-zag scan pattern also supported by the MPEG-2 standard), and AC coefficients having zero magnitude are described in terms of total counts of consecutive zero valued coefficients lying in between two nonzero valued coefficients, in the MPEG-2 coded bit stream. An end-of-block (EOB) code 205 signals the end of the encoded DCT coefficients for the current block. The reduced-quality MPEG-2 coded bit stream 210 includes a DC coefficient 201' identical to the DC coefficient 201 in the original MPEG-2 coded bit stream 200, and a (run, level) event 202' identical to the (run, level) event 202 in the original MPEG-2 coded bit stream 200. Second and third (run, level) events, however, have been omitted from the reduced-quality MPEG-2 bit stream 210, because an EOB code 205' immediately follows the (run, level) event 202'. Therefore, the two nonzero high-order AC DCT coefficients encoded by the second and third (run, level) events 203, 204 have been omitted from the reduced-quality MPEG-2 bit stream 210.

FIG. 13 is a flowchart of a procedure for scaling MPEG-2 coded video using a variety of techniques including the omission of AC DCT coefficients. The procedure operates upon an original-quality MPEG-2 coded video stream by removing AC DCT coefficients in this stream to produce a lower quality MPEG coded video stream. In a first step 221, execution branches to step 222 if the scaled MPEG coded video is to be spatially subsampled. In step 222, the procedure removes any and all DCT coefficients for spatial frequencies in excess of the Nyquist frequency for the downsampled video. For example, if the low-quality video stream will be downsampled by a factor of two in both the vertical and the horizontal directions, then the procedure removes any and all DCT coefficients having a row index (i) greater than four and any and all DCT coefficients having a column index (j) greater than four. This requires the decoding of the (run, level) coded coefficients to the extent necessary to obtain an indication of the coefficient indices. If a sufficient number of the original AC DCT coefficients are removed for a desired bandwidth reduction, then the scaling procedure is finished. Otherwise, execution branches from step 223 to step 224. Execution also continues from step 221 to step 224 if spatial downsampling is not intended.

In step 224, execution branches to step 225 if low-pass scaling is desired. Low-pass scaling requires the least computational resources and may produce the best results if the scaled, low-quality MPEG coded video is spatially downsampled. In step 225, the procedure retains up to a certain number of lowest-order AC DCT coefficients for each block and removes any additional DCT coefficients for each block. This is a kind of frequency domain signal-to-noise ratio scaling (FDSNR) that will be designated FDSNR_LP. A specific example of the procedure for step 225 will be described below with reference to FIG. 14.

Execution continues from step 224 to step 226 if low-pass scaling is not desired. In step 226, execution branches to step 227 if largest magnitude based scaling is desired. Largest magnitude based scaling produces the least squared error or difference between the original-quality MPEG-2 coded video and the reduced-quality MPEG coded video for a given number of nonzero AC coefficients to preserve, but it requires more computational resources than the low-pass scaling of step 225. More computational resources are needed because if there are more nonzero AC coefficients than the desired number of AC coefficients for a block, then the (run, level) events must be decoded fully to obtain the coefficient magnitudes, and additional resources are required to find the largest magnitude coefficients. In step 227, the procedure retains up to a certain number of largest magnitude AC DCT coefficients for each block, and removes any and all additional AC DCT coefficients for each block. This is a kind of frequency domain signal-to-noise ratio scaling (FDSNR) that will be designated FDSNR_LM. A specific example of the procedure for step 227 will be described below with reference to FIG. 15.

If in step 226 largest magnitude based scaling is not desired, then execution continues to step 228. In step 228, execution branches to step 229 to retain up to a certain number of AC DCT coefficients that differ in magnitude from up to that number of largest magnitude AC DCT coefficients by no more than a certain limit. This permits a kind of approximation to FDSNR_LM in which an approximate search is undertaken for the largest magnitude AC DCT coefficients if there are more nonzero AC DCT coefficients than the desired number of AC DCT coefficients in a block. The approximate search can be undertaken using a coefficient magnitude classification technique such as a hashing technique, and the low-pass scaling technique can be applied to the classification level that is incapable of discriminating between the desired number of largest magnitude AC DCT coefficients. A specific example is described below with reference to FIG. 19.

With reference to FIG. 14, there is shown a flowchart of a procedure for scaling MPEG-2 coded video using the low-pass frequency-domain signal-to-noise (FDSNR_LP) scaling technique. This procedure scans and selectively copies components of an input stream of original-quality MPEG-2 coded data to produce an output stream of reduced-quality MPEG-2 coded video. The procedure is successively called, and each call processes coefficient data in the input stream for one 8x8 block of pixels. No more than a selected number "k" of coded lowest order (nonzero or zero valued) AC coefficients are copied for the block where the parameter "k" can be specified for each block.

In a first step 241 of FIG. 14, the procedure parses and copies the stream of original-quality MPEG-2 coded data up to and including the differential DC coefficient variable-length code (VLC). Next, in step 242, a counter variable "l" is set to zero. In step 243, the procedure parses the next (run, level) event VLC in the stream of original-quality MPEG-2 coded data. In step 244, if the VLC just parsed is an end-of-block (EOB) marker, execution branches to step 245 to copy the VLC to the stream of reduced-quality MPEG-2 coded video, and the procedure is finished for the current block.

In step 244, if the VLC just parsed is not an EOB marker, then execution continues to step 246. In step 246, a variable "r" is set equal to the run length of zeroes for the current (run, level) event, in order to compute a new counter value l+r+1. In step 247, if the new counter value l+r+1 is greater than the parameter "k", then the procedure branches to step 248 to copy an EOB marker to the stream of reduced-quality MPEG coded data. After step 248, execution continues to step 249, where the procedure parses the input stream of original-quality MPEG-2 coded data until the end of the next EOB marker, and the procedure is finished for the current block.

In step 247, if the new counter value l+r+1 is not greater than the parameter "k", then execution continues to step 250. In step 250, execution branches to step 251 if the new counter value *l*+r+1 is not equal to "k" (which would be the case if the new counter value is less than "k"). In step 251, the counter state *l* is set equal to the new counter value *l*+r+1. Then, in step 252, the VLC just parsed (which will be a VLC encoding a (run, level) event) is copied from the stream of original-quality MPEG-2 coded data to the stream of reduced-quality MPEG-2 coded data. After step 252, execution loops back to step 243 to continue the scanning of the stream of original-quality MPEG-2 coded data.

In step 250, if the new counter value *l*+r+1 is equal to "k", then execution branches from step 250 to step 253, to copy the VLC just parsed (which will be a VLC encoding a (run, level) event) from the stream of original-quality MPEG-2 coded data to the stream of reduced-quality MPEG-2 coded data. Next, in step 254, the procedure copies an EOB marker to the stream of reduced-quality MPEG-2 coded data. After step 254, execution continues to step 249, where the procedure parses the input stream of original-quality MPEG-2 coded data until the end of the next EOB marker, and the procedure is finished for the current block.

FIG. 15 is a flowchart of a procedure for scaling MPEG-2 coded video using the largest magnitude based frequency-domain signal-to-noise ratio (FDSNR_LM) scaling technique. This routine is successively called, and each call processes coefficient data in the input stream for one 8x8 block of pixels. No more than a specified number "k" of largest magnitude AC DCT coefficients are copied for the block, and a different number "k" can be specified for each block.

In a first step 261 in FIG. 15, the procedure parses and copies the input stream of original-quality MPEG-2 coded data to the output stream of lower-quality MPEG-2 data up to and including the differential DC coefficient variable-length code (VLC). Then in step 262 all (run, level) event VLCs are parsed and decoded until and including the EOB marker of the current block. The decoding produces coefficient identifiers and corresponding quantization indices representing the quantized coefficient values. In step 263, the quantization indices are transformed to quantized coefficient values. In step 264, the (quantized) coefficients are sorted in descending order of their magnitudes. In step 265, the first "k" coefficients of the sorted list are preserved and the last 63-k AC DCT coefficients in the sorted list are set to zero. In step 266, (run, level) event formation and entropy coding (VLC encoding) are applied to the new set of coefficient values. Finally, in step 267, the VLCs resulting from step 266 are copied to the output stream until and including the EOB marker.

The sorting step 264 of the FDSNR_LM procedure can consume considerable computational resources. It is important to notice that not a full sorting of the quantized AC coefficients with respect to their magnitudes but rather a search for a specified number "k" of largest magnitude AC coefficients is all that is required. This task can be performed exactly or approximately in different ways so as to avoid the complexity associated with a conventional sorting procedure. In general, a relatively large number of the 63 AC DCT coefficients will have a quantized value of zero. Only the non-zero coefficients need be included in the sorting process. Moreover, if there are "n" non-zero coefficients and only "k" of them having the largest magnitudes are to be preserved in the output stream, then the sorting process may be terminated immediately after only the largest magnitude "k" coefficients have been found, or equivalently immediately after only the smallest magnitude "n-k" coefficients have been found. Moreover, the sorting procedure itself can be different depending on a comparison of “k” to “n” in order to minimize computations.

With reference to FIG. 16, there is shown a flowchart of a procedure that selects one of a number of techniques for finding a certain number “k” of largest values out of a set of “n” values. In a first step 271, execution branches to step 272 if “k” is less than ½ “n.” In step 272, execution branches to step 273 if “k” is much less than ½ “n.” In step 273, the first “k” values are sorted to produce a list of “k” sorted values, and then the last “n-k” values are scanned for any value greater than the minimum of the sorted “k” values. If a value greater than the minimum of the sorted “k” values is found, then that minimum value is removed and the value greater than the minimum value is inserted into the list of “k” sorted values. At the end of this procedure, the list of sorted “k” values will contain the maximum “k” values out of the original “n” values. A specific example of this procedure is described below with reference to FIG. 17.

In step 272, if “k” is not much less than ½ “n”, then execution branches to step 274. In step 274, a bubble-sort procedure is used, including “k” bottom-up bubble-sort passes over the “n” values to put “k” maximum values on top of a sorting table. An example of such a bubble-sort procedure is listed below:

```
/* TABLE(0) to TABLE(n-1) INCLUDES n VALUES */
/* MOVE THE k LARGEST OF THE n VALUES IN TABLE TO THE RANGE
TABLE(0) TO TABLE(k-1) IN THE TABLE */
/* k <= ½ n */
FOR i=1 to k
FOR j=1 to n-i
IF (TABLE(n-j) > TABLE(n-j-1)) THEN(
        /* SWAP TABLE(n-j) WITH TABLE(n-j-1) */
        TEMP ← TABLE(n-j)
        TABLE(n-j) ← TABLE(n-j-1)
        TABLE(n-j-1) ← TEMP
        NEXT j
        NEXT I
```

In step 271, if "k" is not less than ½ "n", then execution branches to step 275. In step 275, if "k" is much greater than ½ "n", then execution branches to step 276. In step 276, a procedure similar to step 273 is used, except the "n-k" minimum values are maintained in a sorted list, instead of the "k" maximum values. In step 276, the last "n-k" values are placed in the sort list and sorted, and then the first "k" values are scanned for any value less than the maximum value in the sorted list. If a value less than the maximum value in the sorted list is found, then the maximum value in the sorted list is removed, and the value less than this maximum value is inserted into the sorted list. At the end of this procedure, the values in the sorted list are the "n-k" smallest values, and the "k" values excluded from the sorted list are the "k" largest values.

In step 275, if "k" is not much greater than ½ "n", then execution branches to step 277. In step 277, a bubble-sort procedure is used, including "n-k" top-down bubble-sort passes over the "n" values to put "n-k" minimum values at the bottom of a sorting table.

Consequently, the k maximum values will appear in the top "k" entries of the table. An example of such a bubble-sort procedure is listed below:

```
/* TABLE(0) to TABLE(n-1) INCLUDES n VALUES */
/* MOVE THE n-k SMALLEST OF THE n VALUES IN THE TABLE */
/* TO THE RANGE TABLE(k) TO TABLE(n-1) IN THE TABLE */
/* n > k >= ½ n */
FOR i=1 to n-k
FOR j=0 to n-i-1
IF (TABLE(j) < TABLE(j+1)) THEN(
        /* SWAP TABLE(j) WITH TABLE(j+1)*/
        TEMP ←TABLE(j)
        TABLE(j) ← TABLE(j+1)
        TABLE(j+1) ← TEMP
        NEXT j
        NEXT I
```

Turning now to FIG. 17, there is shown a flowchart of a procedure for finding up to a specified number "k" of largest magnitude AC DCT coefficients from a set of "n" coefficients, corresponding to the procedure of FIG. 16 for the case of k << ½n. In a first step 281, a counter "i" is set to zero. In step 282, the next AC DCT coefficient is obtained from the input stream of original-quality MPEG-2 coded data. If an EOB marker is reached, as tested in step 283, then execution returns. In step 284, the counter "i" is compared to the specified number "k", and if "i" is less than "k", execution continues to step 285. In step 285, a coefficient index and magnitude for the AC DCT coefficient is placed on a sort list. In step 286, the counter "i" is incremented, and execution loops back to step 282.

Once the sort list has been loaded with indices and magnitudes for "k" AC DCT coefficients and one additional coefficient has been obtained from the input stream, execution branches from step 284 to step 287. In step 287 the list is sorted by magnitude, so that the minimum magnitude appears at the end of the list. Then in step 288 the coefficient magnitude of the current coefficient last obtained from the input stream is compared to the magnitude at the end of the list. If the coefficient magnitude of the current coefficient is not greater than the magnitude appearing at the end of the list, then execution continues to step 289 to get the next AC DCT coefficient from the input stream. If an EOB marker is reached, as tested in step 290, then execution returns. Otherwise, execution loops back to step 288.

In step 288, if the magnitude of the current coefficient is greater than the magnitude at the end of the list, then execution branches to step 291. In step 291, the entry at the end of the list is removed. In step 292, a binary search is performed to determine the rank position of the magnitude of the current coefficient, and in step 293, the current coefficient index and magnitude are inserted into the list at the rank position. The list, for example, is a linked list in the conventional fashion to facilitate the insertion of an entry for the current coefficient at any position in the list. After step 293, execution loops back to step 288.

An approximation technique of coefficient magnitude classification can be used to reduce the computational burden of sorting by coefficient magnitude. A specific example is the use of hashing of the coefficient magnitude and maintaining lists of the indices of coefficients having the same magnitude classifications. As shown in FIG. 18, a hash table 300 is linked to hash lists 301 storing the indices of classified coefficients. As shown, the hash table 300 is a list of 2^M entries, where "M" is three, and an entry has a value of zero if its associated list is empty, and otherwise the entry has a pointer to the end of the coefficients in its associated list. The lists shown in FIG. 18 have fixed memory allocations in which the pointers in the hash table also indicate the number of coefficient indices in the respective hash lists. Alternatively, the hash lists could be dynamically allocated and linked in the conventional fashion.

FIG. 19 shows a flowchart of a procedure for using the hash table 300 and hash lists 301 of FIG. 18 to perform a sort of "k" coefficients having approximately the largest magnitudes from a set of "n" coefficients. This approximation technique ensures that none of the "k" coefficients selected will have a magnitude that differs by more than a certain error limit from the smallest magnitude value of "k" coefficients having the largest magnitude. The error limit is established by the number of hash table entries, and it is the range of the magnitudes that can be hashed to the same hash table entry.

In a first step 311 in FIG. 19, the hash table is cleared. Then in step 312, the next AC DCT coefficient is obtained from the input stream. If an EOB marker is not reached, as tested in step 313, then execution continues to step 314. In step 314, a hash table index is stripped from the most significant bits (MSBs) of the coefficient magnitude. For the hash table in FIG. 18 having eight entries, the three most significant bits of the coefficient magnitude are stripped from the coefficient magnitude. This is done by a bit masking operation together with a logical arithmetic shift operation. Then in step 315, the coefficient index is inserted on the hash list of the indexed hash table entry. For example, the hash table entry is indexed to find the pointer to where the coefficient index should be inserted, and then the pointer in the hash table entry is incremented. After step 315, execution loops back to step 312. Once all of the AC coefficients for the block have been classified by inserting them in the appropriate hash lists, an EOB marker will be reached, and execution will branch from step 313 to step 316.

Beginning in step 316, the hash table and hash lists are scanned to find approximately the "k" largest magnitude coefficients. The hash lists linked to the bottom entries of the hash table will have the indices for the largest magnitude coefficients. Each hash list is scanned from its first entry to its last entry, so that each hash list is accessed as a first-in-first-out queue. Therefore, in each magnitude classification, the coefficient ordering in the output stream will be the same as the coefficient ordering in the input stream, and the approximation will have a "low pass" effect in which possibly some lower-frequency coefficients having slightly smaller magnitudes will be retained at the expense of discarding some higher-frequency coefficients having slightly larger magnitudes. (The approximation results from the fact that the last hash list to be scanned is not itself sorted, and to eliminate the error of the approximation, the last hash list to be scanned could be sorted.)

In step 316, a scan index "i" is set to 2^M-1 in order to index the hash table beginning at the bottom of the table, and a counter "j" is set equal to "k" in order to stop the scanning process after finding "k" coefficients. Next, in step 317, the hash table is indexed with "i". In step 318, if the indexed entry of the hash table is zero, then execution branches to step 319. In step 319, the procedure is finished if "i" is equal to zero; otherwise, execution continues to step 320. In step 320, the index "i" is decremented, and execution loops back to step 317.

If in step 318 the indexed hash table entry is not zero, then execution continues to step 321. In step 321, the next entry is obtained from the indexed hash list, and the coefficient index in the entry is used to put the indexed coefficient in the output stream. Then in step 322 execution branches to step 319 if the end of the indexed hash list is reached in the previous step 321. If the end of the list was not reached in step 321, then execution continues from step 322 to step 323. In step 323 the counter "j" is decremented, and in step 324 the counter "j" is compared to zero. In step 324, if the counter "j" is less than or equal to zero, then the procedure is finished. Otherwise, if the counter "j" is not less than or equal to zero in step 324, execution loops back to step 321.

The FDSNR_LM procedure, as described above, in general provides a significant improvement in peak signal-to-noise ratio (PSNR) over the FDSNR_LP procedure when each procedure retains the same number of non-zero AC DCT coefficients. It has been found, however, that substantially more bits are required for the (run, level) coding of the non-zero AC DCT coefficients resulting from the FDSNR_LM procedure than those resulting from the FDSNR_LP procedure, provided that the same coefficient quantization and scanning method is used. Therefore, the FDSNR_LM procedure provides at best a marginal improvement in rate-distortion (PSNR as a function of bit rate) over the FDSNR_LP procedure unless the non-zero AC DCT coefficients for the FDSNR_LM procedure are quantized, scanned, and/or (run, level) coded in a fashion different from the quantization, scanning, and/or (run, level) coding of the coefficients in the original MPEG-2 clip. A study of this problem resulted in a discovery that it is sometimes possible to reduce the number of bits for (run, level) coding of coefficients for an 8x8 block including a given number of the non-zero largest magnitude AC DCT coefficients if additional coefficients are also (run, level) coded for the block.

The (run, level) coding of the non-zero AC DCT coefficients from the FDSNR_LM procedure has been found to require more bits than from the FDSNR_LP procedure due to an increased occurrence frequency of escape sequences for the (run, level) coding. The increased frequency of escape sequences is an indication that the statistical likelihood of possible (run, level) combinations for the non-zero AC DCT coefficients selected by the FDSNR_LM procedure is different from the statistical likelihood of possible (run, level) combinations for the non-zero AC DCT coefficients produced by the standard MPEG-2 coding process and in particular those selected by the FDSNR_LP procedure.

The MPEG-2 coding scheme assigns special symbols to the (run, level) combinations that occur very frequently in ordinary MPEG-2 coded video. The most frequent (run, level) combinations occur for short run lengths (within the range of about 0 to 5, where the run length can range from 0 to 63) and relatively low levels (about 1 to 10, where the level can range from 1 to 2048). The most frequent of these special symbols are assigned the shortest variable-length code words (VLCs). If a (run, level) combination does not have such a special symbol, then it is coded as an escape sequence including a 6-bit escape sequence header code word followed by a 6-bit run length followed by a 12 bit signed level. An escape sequence requires a much greater number of bits than the special symbols, which have varying lengths depending on their relative frequency. In particular, the escape sequences each have 24 bits, and the special symbols have variable-length code words having a maximum of 17 bits.

There are two (run, level) VLC tables. The first coding table is designated TABLE 0, and the second is designated TABLE 1. These tables specify the (run, level) combinations having special symbols, and the special symbol for each such combination. For each table, the (run, level) combinations having special symbols, and the range of the VLC bit lengths of the special symbols, are summarized below:

SUMMARY OF PROPERTIES OF DCT COEFFICIENT TABLE ZERO

(Table Zero is Table B.14, p. 135 of ISO/IEC 13818-2 1996E)

Run	Range of Levels	Range of Code Lengths
0	1 to 40	2 to 16
1	1 to 18	4 to 17
2	1 to 5	5 to 14
3	1 to 4	6 to 14
4	1 to 3	6 to 13
5	1 to 3	7 to 14
6	1 to 3	7 to 17
7	1 to 2	7 to 13
8	1 to 2	8 to 13
9	1 to 2	8 to 14

10	1 to 2	9 to 14
11	1 to 2	9 to 17
12	1 to 2	9 to 17
13	1 to 2	9 to 17
14	1 to 2	11 to 17
15	1 to 2	11 to 17
16	1 to 2	11 to 17
17	1	13
18	1	13
19	1	13
20	1	13
21	1	13
22	1	14
23	1	14
24	1	14
25	1	14
26	1	14
27	1	17
28	1	17
29	1	17
30	1	17
31	1	17

SUMMARY OF PROPERTIES OF DCT COEFFICIENT TABLE ONE

(Table One is Table B.15, p. 139 of ISO/IEC 13818-2 1996E)

Run	Range of Levels	Range of Code Lengths
0	1 to 40	3 to 16
1	1 to 18	4 to 17
2	1 to 5	6 to 14
3	1 to 4	6 to 14
4	1 to 3	7 to 13
5	1 to 3	7 to 14
6	1 to 3	8 to 17
7	1 to 2	8 to 13
8	1 to 2	8 to 13
9	1 to 2	8 to 14
10	1 to 2	8 to 14
11	1 to 2	9 to 17
12	1 to 2	9 to 17
13	1 to 2	9 to 17
14	1 to 2	10 to 17
15	1 to 2	10 to 17
16	1 to 2	11 to 17
17	1	13
18	1	13

19	1	13
20	1	13
21	1	13
22	1	14
23	1	14
24	1	14
25	1	14
26	1	14
27	1	17
28	1	17
29	1	17
30	1	17
31	1	17

The FDSNR_LP procedure selected AC DCT coefficients have (run, level) symbol statistics that are similar to the statistics of ordinary MPEG-2 coded video, and therefore the FDSNR_LP AC DCT coefficients have a similar frequency of occurrence for escape sequences in comparison to the ordinary MPEG-2 coded video. In contrast, the FDSNR_LM procedure selects AC DCT coefficients resulting in (run, level) combinations that are less likely than the combinations for ordinary MPEG-2 coded video. This is due to two reasons. First, the FDSNR_LM procedure selects AC DCT coefficients having the highest levels. Second, the FDSNR_LM procedure introduces higher run lengths due to the elimination of coefficients over the entire range of coefficient indices. The result is a significantly increased rate of occurrence for escape sequences. Escape sequences form the most inefficient mode of coefficient information encoding in MPEG-2 incorporated into the standard so as to cover important but very rarely occurring coefficient information.

In order to improve the rate-distortion performance of the scaled-quality MPEG-2 coded video from the FDSNR_LM procedure, the non-zero AC DCT coefficients selected by the FDSNR_LM procedure should be quantized, scanned, and/or (run, level) coded in such a way that tends to reduce the frequency of the escape sequences. For example, if the original-quality MPEG-2 coded video was (run, level) coded using TABLE 0, then the largest magnitude coefficients should be re-coded using TABLE 1 because TABLE 1 provides shorter length VLCs for some (run, level) combinations having higher run lengths and higher levels. It is also possible that re-coding using the alternate scan method instead of the zig-zag scan method may result in a lower frequency of occurrence for escape sequences. For example, each picture could be (run, level) coded for both zig-zag scanning and alternate scanning, and the scanning method providing the fewest escape sequences, or the least number of bits total, could be selected for the coding of the reduced-quality coded MPEG video.

There are two methods having general applicability for reducing the frequency of escape sequences resulting from the FDSNR_LM procedure. The first method is to introduce a non-zero, "non-qualifying" AC DCT coefficient of the 8x8 block into the list of non-zero qualifying AC DCT coefficients to be coded for the block. In this context, a "qualifying" coefficient is one of the k largest magnitude coefficients selected by the FDSNR_LM procedure. The non-qualifying coefficient referred to above, must be lying in between two qualifying AC DCT coefficients (in the coefficient scanning order) that generate the (run, level) combination causing the escape sequence. Moreover, this non-qualifying coefficient must cause the escape sequence to be replaced with two shorter length VLCs when the AC DCT coefficients are (run, level) coded. This first method has the effect of not only decreasing the number of bits in the coded reduced-quality MPEG video in most cases, but also increasing the PSNR.

The qualifying AC DCT coefficient causing the escape sequence that is first in the coefficient scanning order will be simply referred to as the first qualifying coefficient. The qualifying AC DCT coefficient causing the escape sequence that is second in the coefficient scanning order will be simply referred to as the second qualifying coefficient. For example, suppose the qualifying coefficients in zig-zag scan order for an 8x8 block include C_{51} followed by C_{15} having a level of 40. If only the qualifying coefficients were (run, level) coded for the microblock, C_{15} would result in a run length of 3, because there are a total of three non-qualifying coefficients (C_{42}, C_{33}, and C_{24}) between C_{51} and C_{15} in the scan order. Therefore, C_{15} would have to be coded as an escape sequence, because a run of 3 and level of 40 does not have a special symbol. In this example, the escape sequence is in effect caused by a first qualifying coefficient, which is C_{51}, and a second qualifying coefficient, which is C_{15}. This escape sequence can possibly be eliminated say, if C_{24} is a non-zero, non-qualifying coefficient of the block, C_{24} has a level of 5 or less, and C_{24} is (run, level) coded together with the qualifying coefficients. For example, assuming that C_{24} has a level of 5, and using the MPEG-2 (run, level) coding TABLE 1, then C_{24} has a run length of two and is coded as the special symbol 0000 0000 1010 0s, where "s" is a sign bit, and C_{15} now has a run length of 0 and is coded as the special symbol 0000 0000 0010 00s. Such a consideration clearly applies to the rest of the non-zero non-qualifying coefficients lying in between the two qualifying coefficients producing the escape sequence. In the above example, these non-qualifying coefficients are C_{42} and C_{33}.

Whether or not an escape sequence can be eliminated from the (run, level) coding of the qualifying coefficients can be determined by testing a sequence of conditions. The first condition is that the second qualifying coefficient must have a level that is not greater than the maximum level of 40 for the special (run, level) symbols. If this condition is satisfied, then there must be a non-zero non-qualifying AC DCT coefficient that is between the first and second qualifying coefficients in the coefficient scanning order. If there is such a non-qualifying coefficient, then the combination of its level and the run length between the first qualifying coefficient and the non-qualifying coefficient in the coefficient scanning order must be one of the special (run, level) symbols. If so, then the combination of the level of the second qualifying coefficient and the run length between the non-qualifying coefficient and the second qualifying coefficient must also be a special (run, level) symbol, and if so, all required conditions have been satisfied. If not, then the conditions with respect to the non-qualifying coefficient are successively applied to any other non-zero non-qualifying AC DCT coefficient of the block lying in between the two qualifying coefficients, until either all conditions are found to be satisfied or all such non-qualifying coefficients are tested and failed. If there are sufficient computational resources, this search procedure should be continued to find all such non-qualifying coefficients that would eliminate the escape sequence, and to select the non-qualifying coefficient that converts the escape sequence to the pair of special symbols having respective code words that in combination have the shortest length.

A flow chart for a modified FDSNR_LM procedure using the first method is shown in FIGS. 20 and 21. In a first step 331 of FIG. 20, the procedure finds up to "k" largest magnitude non-zero AC DCT coefficients (i.e., the "qualifying coefficients") for the block. (This first step 331 is similar to steps 261 to 265 of FIG. 15, as described above.) In step 332, (run, level) coding of the qualifying coefficients is begun in the scan order using the second coding table (Table 1). This (run, level) coding continues until an escape sequence is reached in step 333, or the end of the block is reached in step 336. If an escape sequence is reached, execution branches from step 333 to step 334. If the level of the second qualifying coefficient causing the escape sequence is greater than 40, execution continues from step 334 to step 336. Otherwise, execution branches from step 334 to step 335 to invoke a subroutine (as further described below with reference to FIG. 21) to possibly include a non-zero non-qualifying AC DCT coefficient in the (run, level) coding to eliminate the escape sequence. The subroutine either returns without success, or returns such a non-qualifying coefficient so that the escape sequence is replaced with the two new (run, level) codings of the first qualifying coefficient and the non-qualifying coefficient and then the non-qualifying coefficient and the second qualifying coefficient. From step 335, execution continues to step 336. Execution returns from step 336 if the end of the block is reached. Otherwise, execution continues from step 336 to step 337, to continue (run, level) coding of the qualifying coefficients in the scan order using the second coding table (TABLE 1). This (run, level) coding continues until an escape sequence results, as tested in step 333, or until the end of the block is reached, as tested in step 336.

With reference to FIG. 21, there is shown a flow chart of the subroutine (that was called in step 335 of FIG. 20) for attempting to find a non-zero, non-qualifying AC DCT coefficient that can be (run, level) coded to eliminate an escape sequence for a qualifying coefficient. In a first step 341, the procedure identifies the first qualifying coefficient and the second qualifying coefficient causing the escape sequence. For example, the subroutine of FIG. 21 can be programmed as a function having, as parameters, a pointer to a list of the non-zero AC DCT coefficients in the scan order, an index to the first qualifying coefficient in the list, and an index to the second qualifying coefficient in the list. In step 342, the subroutine looks for a non-zero non-qualifying AC DCT coefficient between the first and the second qualifying coefficients in the scan order. For example, the value of the index to the first qualifying coefficient is incremented and compared to the value of the index for the second qualifying coefficient, and if they are the same, there is no such non-qualifying coefficient. Otherwise, if the new coefficient pointed to (by incrementing the index of the first qualifying coefficient) is a non-zero coefficient then it becomes a candidate non-qualifying coefficient deserving further testing. If however the new coefficient pointed to (by incrementing the index of the first qualifying coefficient) has a value zero then it is not a candidate non-qualifying coefficient. If no such (candidate) non-qualifying coefficients are found, as tested in step 343, then execution returns from the subroutine with a return code indicating that the search has been unsuccessful. Otherwise, execution continues to step 344.

In step 344, the non-qualifying coefficient is (run, level) coded, to determine in step 345 whether it codes to an escape sequence. If it codes to an escape sequence, then execution loops back from step 345 to step 342 to look for another non-zero non-qualifying AC DCT coefficient in the scan order between the first and second qualifying coefficients. If it does not code to an escape sequence, then execution continues from step 345 to step 346. In step 346, the second qualifying coefficient is (run, level) coded, using the new run length, which is the number of coefficients in the scan order between the non-qualifying coefficient and the second qualifying coefficient. If it codes to an escape sequence, as tested in step 347, then execution loops back from step 347 to step 342 to look for another non-zero non-qualifying AC DCT coefficient in the scan order between the first and second qualifying coefficients. If it does not code to an escape sequence, then execution continues from step 347 to step 348.

In step 348, execution returns with a successful search result unless a continue search option has been selected. If the continue search option has been selected, then execution branches from step 348 to step 349 to search for additional non-zero non-qualifying AC DCT coefficients that would eliminate the escape sequence. In other words, steps 342 to 347 are repeated in an attempt to find additional non-zero non-qualifying AC DCT coefficients that would eliminate the escape sequence. If no more such non-qualifying coefficients are found, as tested in step 350, execution returns with a successful search result. Otherwise, execution branches from step 350 to step 351 to select the non-qualifying coefficient giving the shortest overall code word length and/or the largest magnitude for the best PSNR, and execution returns with a successful search result. For example, for each non-qualifying coefficient that would eliminate the escape sequence, the total bit length is computed for the (run, level) coding of the non-qualifying coefficient and the second qualifying coefficient. Then a search is made for the non-qualifying coefficient producing the shortest total bit length, and if two non-qualifying coefficients which produce the same total bit length are found, then the one having the largest level is selected for the elimination of the escape sequence.

A second method of reducing the frequency of occurrence of the escape sequences in the (run, level) coding of largest magnitude AC DCT coefficients for an 8x8 block is to change the mapping of coefficient magnitudes to the levels so as to reduce the levels. Reduction of the levels increases the likelihood that the (run, level) combinations will have special symbols and therefore will not generate escape sequences. This second method has the potential of achieving a greater reduction in bit rate than the first method, because each escape sequence can now be replaced by the codeword for one special symbol, rather than by the two codewords as is the case for the first method. The second method, however, may reduce the PSNR due to increased quantization noise resulting from the process producing the lower levels. Therefore, if a desired reduction of escape sequences can be achieved using the first method, then there is no need to perform the second method, which is likely to reduce the PSNR. If the first method is used but not all of the escape sequences have been eliminated, then the second method could be used to possibly eliminate the remaining escape sequences.

The mapping of coefficient magnitudes to the levels can be changed by decoding the levels to coefficient magnitudes, changing the quantization scale factor (qsi), and then re-coding the levels in accordance with the new quantization scale factor (qsi). The quantization scale factor is initialized in each slice header and can also be updated in the macroblock header on a macroblock basis. Therefore it is a constant for all blocks in the same macroblock. In particular, the quantization scale factor is a function of a q_scale_type parameter and a quantizer_scale_code parameter. If q_scale_type = 0, then the quantizer scale factor (qsi) is twice the value of q_scale_code. If q_scale_type =1, then the quantizer scale factor (qsi) is given by the following table, which is the right half of Table 7-6 on page 70 of ISO/IEC 13838-2:1996(E):

quantizer_scale_code	quantization scale factor (qsi)
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	10
10	12
11	14
12	16
13	18
14	20
15	22

16	24
17	28
18	32
19	36
20	40
21	44
22	48
23	52
24	56
25	64
26	72
27	80
28	88
29	96
30	104
31	112

In a preferred implementation, to reduce the coefficient levels, the quantization scale factor is increased by a factor of two, and the levels of the non-zero AC DCT coefficients are reduced by a factor of two, so long as the original value of the quantization scale factor is less than or equal to one-half of the maximum possible quantization scale factor. For q_scale_type = 1, a factor of two increase in the quantization scale factor (qsi) is most easily performed by a table lookup of a new quantization_scale_code using the following conversion table:

Original quantization_scale_code	New quaitization_scale_code
1	2
2	4
3	6
4	8
5	9
6	10
7	11
8	12
9	14
10	16
11	17
12	18
13	19
14	20
15	21
16	22
17	24
18	25
19	26

20	27
21	28
22	29
23	30
24	31

In a preferred method for generation of trick mode files, the quantization scale factor is adjusted in order to achieve a desired reduction in the escape sequence occurrence frequency resulting from the modified FDSNR_LM procedure, and the number (k) of largest magnitude coefficients is adjusted in order to achieve a desired reduction in bit rate. A specific implementation is shown in the flow chart of FIGS. 22-23. In a first step 361, the number (k) of largest magnitude AC coefficients per 8x8 block is initially set to a value of 9, and the quantization scaling factor (QSF) is initially set to a value of 2. Then conversion of the I frames of an original-quality MPEG-2 coded video clip to a lower quality level begins. When a picture header is encountered in step 362, indicating the beginning of a new I frame, execution continues to step 363. In step 363, execution branches depending on the value of the intra_vlc_format parameter in the picture header of the original-quality MPEG-2 coded video clip. This value is either 0, indicating that the first (run, level) coding table (TABLE 0) was used for coding the picture, or 1, indicating that the second (run, level) coding table (TABLE 1) was used for coding the picture. In either case, the down scaled quality picture will be coded with the second (run, level) coding table. If the intra_vlc_format parameter is equal to 0 execution continues from step 363 to step 364 where TABLE 0 is read in for (run, level) symbol decoding in the original-quality MPEG-2 coded clip. Otherwise, if the intra_vlc_format parameter is equal to 1, then execution continues from step 363 to step 365 where TABLE 1 is read in for (run, level) symbol decoding in the original-quality MPEG-2 coded clip.

After steps 364 and 365, execution continues to step 366. In step 366, the modified FDSNRS_LM procedure is applied to the 8x8 blocks of the current slice, using the adjusted quantization scale index, if the adjusted quantization scale index is less than the maximum possible quantization scale index. In step 367, execution loops back to step 362 to continue 8x8 block conversion until a new slice header is encountered, indicating the beginning of a new slice. Once a new slice is encountered, execution continues from step 367 to step 368. In step 368, the average escape sequence occurrence frequency per block for the last slice is compared to a threshold TH1. If the escape sequence occurrence frequency is greater than the threshold, then execution branches to step 369. In step 369, if the quantization scaling factor (QSF) is less than or equal to a limit value such as 2, then execution branches to step 370 to increase the quantization scaling factor (QSF) by a factor of two.

In step 368, if the escape sequence occurrence frequency is not greater than the threshold TH1, then execution continues to step 371 of FIG. 23. In step 371, the average escape sequence occurrence frequency per 8x8 block for the last slice is compared to a threshold TH2. If the escape sequence occurrence frequency is less than the threshold TH2, then execution branches to step 372. In step 372, if the quantization scaling factor (QSF) is greater than or equal to a limit value such as 2, then execution branches to step 373 to decrease the quantization scaling factor (QSF) by a factor of two. After step 373, and also after step 370 of FIG. 22, execution continues to step 374 of FIG. 23. In step 374, execution continues to step 375 if a backtrack option has been selected. In step 375, re-coding for the last slice is attempted using the adjusted quantization scale factor. The new coding, or the coding that gives the best results in terms of the desired reduction of escape sequence occurrence frequency, is selected for use in the scaled quality picture. After step 375, execution continues to step 376. Execution also continues to step 376 from: step 369 in FIG 22 if the quantization scaling factor (QSF) is not less than or equal to 2; step 371 in FIG 23 if the escape sequence occurrence frequency is not less than the threshold TH2; step 372 in FIG 23 if the quantization scaling factor (QSF) is not greater than or equal to 2; and from step 374 in FIG 23 if the backtrack option has not been selected.

In step 376, the average bit rate of the (run, level) coding per 8x8 block for at least the last slice is compared to a high threshold TH3. Preferably this average bit rate is a running average over the already processed segment of the current scaled quality I-frame, and the high threshold TH3 is selected to prevent video buffer overflow in accordance with the MPEG-2 Video Buffer Verifier restrictions. If the average bit rate exceeds the high threshold TH3, then execution continues to step 377, where the number (k) of non-zero largest magnitude AC coefficients per 8x8 block is compared to a lower limit value such as 6. If the number (k) is greater than or equal to 6, then execution continues to step 378 to decrement the number (k).

In step 376, if the average bit rate is not greater than the threshold TH3, then execution continues to step 379. In step 379, the average bit rate is compared to a lower threshold TH4. If the average bit rate is less than the threshold TH4, then execution branches from step 379 to step 380, where the number (k) of non-zero largest magnitude AC DCT coefficients per 8x8 block is compared to a limit value of 13. If the number (k) is less than or equal to 13, then execution continues to step 381 to increment the number (k). After step 378 or 381, execution continues to step 382. In step 382, execution continues to step 383 if a backtrack option is selected. In step 383, an attempt is made to re-code the last slice for the scaled quality picture using the adjusted value of the number (k) of non-zero largest magnitude AC DCT coefficients per block. After step 383, execution loops back to step 362 of FIG. 22 to continue generation of the scaled quality clip. Execution also loops back to step 362 of FIG. 22 after: step 377 if the value of (k) is not greater than or equal to 6; step 379 if the average bit rate is not less than the threshold TH4; step 380 if the value of (k) is not less than or equal to 13; and step 382 if the backtrack option has not been selected. Coding of the scaled quality clip continues until the end of the original quality clip is reached in step 364 of FIG. 22, in which case execution returns.

In a preferred implementation, a fast forward trick mode file and a fast reverse trick mode file are produced from an original-quality MPEG-2 coded video main file when the main file is ingested into the video file server. As shown in FIG. 24, a volume generally designated 390 is allocated to store the main file 391. The volume 390 includes an allocated amount of storage that exceeds the real file size of the main file 391 in order to provide additional storage for meta-data 392, the fast forward trick file 393, and the fast reverse trick file 394. The trick files are not directly accessible to clients as files; instead, the clients may access them thorough trick-mode video service functions. With this strategy, the impact on the asset management is a minimum. No modification is needed for delete or rename functions.

Because the volume allocation is done once for the main file and its fast forward and fast reverse trick mode files, there is no risk of lack of disk space for production of the trick files. The amount of disk blocks to allocate for these files is computed by the video service using a volume parameter (vsparams) specifying the percentage of size to allocate for trick files. A new encoding type is created in addition to types RAW for direct access and MPEG2 for access to the main file. The new encoding type is called EMPEG2, for extended MPEG2, for reference to the main file plus the trick files. The video service allocates the extra file size only for these files.

For the transfer of these files to archive or to another video file server, it would be useful to transfer all the data even if it is a non-standard format. For the FTP copy-in, a new option is added to specify if the source is in the EMPEG2 format or if it is a standard MPEG2 file. In the first case, the copy-in should provide the complete file 390. In the second case, the video service allocates the extra size and the processing is the same as for a record. For the copy-out, the same option can be used to export the complete file 390 or only the main part 391. The archiving is always done on the complete file 390.

The trick mode file production is done by a new video service procedure. This procedure takes as input the speed-up factor (or the target trick mode file size) along with the number of freeze (P or B) frames to insert in between the scaled I frames and then generates both the fast forward file 393 and the fast reverse file 394 for this speed-up factor (or target trick mode file size) and with the specified number of interleaving freeze frames. Since the bandwidth of the original clip (in the main file) and the bandwidths of the two trick mode clips (in the fast forward and fast reverse files) are the same, the speed-up factor and the target trick mode file size are equivalent pieces of information. A default speed-up factor (system parameter) can be used. The main file is read and the trick mode files are produced. If a trick mode file already exists with the same speed-up factor, it is rewritten or nothing is done depending on an option. Multiple trick mode files could be created with different speed-up factors. But it is preferred to permit only one set of fast forward and fast reverse trick mode files to be produced at a time (i.e., no parallel generation with different speed-up factors). The current speed-up factor is a parameter of the volume parameters (vsparams).

As stated above another parameter to be provided to the video service procedure in charge of trick mode file generation is the number of freeze frames to be inserted in between consequent scaled I frames. The preferred values for this parameter are 0 and 1, although other positive integer values greater than 1 are also possible. The inclusion of freeze frames due to their very small sizes spare some bandwidth which can then be used to improve the quality of scaled I frames. Hence, the freeze frames in this context provide a mechanism to achieve a trade-off between the scaled I frame quality and the temporal (motion) sampling. Depending on the speed-up factor (or the target trick mode file size) and also the number of interleaving freeze frames to be inserted, the video service procedure in charge of trick mode file generation determines a sub-sampling pattern (closest to uniform) to choose the original I frames which will be scaled and included in the trick mode files. For example, the case of an original clip with 10 frames per GOP, a trick mode file size which is 10% of the main file together with 0 freeze frames, implies the use of all original I frames for being scaled and included in the trick mode file. This will typically result in a low quality scaling. As another example, the case of an original clip with 10 frames per GOP, a trick mode file size which is 10% of the main file together with 1 freeze frame, implies the use of a 2 to 1 (2:1) sub-sampling on the original I frames which will choose every other original I frame for being scaled and included in the trick mode file.

FIG. 25 is a more detailed diagram of the volume 390, showing additional meta-data and related data structures. The Inode 401 includes 4 disk blocks containing a file-system oriented description of the file. The Meta-data (MD) directory 402 includes 4 disk blocks describing each entry of the meta-data area 392. The entries of the meta-data area 392 include a description of the MPEG-2 meta-data 403, a description of the trick files header meta-data 404, and a description of the GOP index meta-data 405. The MPEG-2 meta-data 403 includes 15 disk blocks maximum.

The trick files header 404 includes 1 disk block, which specifies the beginning of free area (end of last trick file) in blocks, the number of trick files couple (FF FR), and for each trick file, a speed-up factor, a block address of the GOP index, a block address of the trick file forward, a byte length of the trick file forward, a block address of the trick file reverse, a byte length of the trick file reverse, a frames number of the trick file, and a number of GOP of each trick files.

The GOP index includes 2024 disk blocks. The GOP index specifies, for each GOP, a frame number, a pointer to the MPEG-2 data for the GOP in the main file, and various flags and other attributes of the GOP. The flags indicate whether the GOP entry is valid and whether the GOP is open or closed. The other attributes of the GOP include the maximum bit rate, the average bit rate, the AAU size in bytes, the APU duration in seconds, the audio PES packet starting locations, the AAU starting locations, the AAU PTS values, and the decode time stamp (DTS) and the value of the program clock reference (PCR) extrapolated to the first frame of the GOP. The size of all the data preceding the main file is, for example, 1 megabyte.

There is one GOP index 406 for both the fast forward file 393 and the fast reverse file 394. The GOP index 406 of the trick files is different than the GOP index 405 of the main file. The GOP index 406 of the trick files contains, for each GOP, the byte offset in the trick file forward of the TS packet containing the first byte of the SEQ header, the frame number in the fast forward file of the GOP (the same value for the fast reverse file can be computed from this value for the fast forward file), the frame number in the original file of the first frame of the GOP, and the byte offset in the original file of the same frame (to resume after fast forward or reverse without reading the main GOP index).

The GOP index 405 for the main file and the GOP index 406 for the fast forward and fast reverse trick files provides a means for rapidly switching between the normal video-on-demand play operation during the reading of the main file, and the fast-forward play during the reading of the fast-forward file, and the fast-reverse play during the reading of the fast reverse file. For example, FIG. 26A illustrates the read access to various GOPs in the main file, fast forward file, and fast reverse file, during a play sequence listed in FIG. 26B. Due to the presence of down-scaled I frames and possibly consequent freeze frames in the trick mode files, the video buffer verifier (VBV) model for a trick mode file is different than the VBV model of the main file. Consequently, the mean video decoder main buffer fullness levels can be significantly different for these files. For example, a transition from the main file to one of the trick files will usually involve a discontinuity in the mean video decoder main buffer fullness level, because only the I frames of the main file correspond to frames in the trick files, and the corresponding I frames have different bit rates when the trick mode I frames are scaled down for a reduced bit rate. An instantaneous transition from a trick file back to the main file may also involve a discontinuity especially when freeze frames are inserted between the I frames for trick mode operation. To avoid these discontinuities, the seamless splicing procedure of FIGS. 3 to 6 as described above is used during the transitions from regular play mode into trick mode and similarly from trick mode back into the regular play mode. Through the use of the seamless splicing procedure to modify the video stream content, for example for the "Seamless Splice" locations identified in FIG. 26A, the video decoder main buffer level will be managed so as to avoid both overflows and underflows leading to visual artifacts.

It is desired to copy in and out of the volume 390 with or without the meta-data 392 and the trick files 393, 394. This is useful to export and/or import complete files without regenerating the trick files. The file encoding type is now recognized as a part of the volume name. Therefore there can be multiple kinds of access to these files. The read and write operations are done by derivations of the class file system input/output (FSIO) which takes into account the proper block offset of the data to read or write. There is one derivation of FSIO per encoding type, providing three different access modes. EMGP3, MPEG2, and RAW. EMPEG2 accesses the whole volume from the beginning of the meta-data array, and in fact provides access to the entire volume except the inode 401, but no processing is done. MPEG2 access only the main part of the asset with MPEG processing, including file analyze and meta-data generation in a write access. RAW access only the main part of the asset without processing. These access modes are operative for read and write operations for various access functions as further shown in FIG. 27.

During a record operation, the video service allocates a volume and computes the number of block to allocate using the volume parameter giving the percentage to add for the trick files. Then, the size in blocks given to the stream server is the main part size only without the extension for the trick files. This avoids using the reserved part of the volume when the effective bit rate is higher than the requested bit rate. At the end of a record operation or a FTP copyin operation, the video service calls a procedure CMSPROC_GETATTR, and the stream server returns the actual number of bytes received and the actual number of blocks used by the main file plus the meta-data. The same values are returned for both MPEG2 and EMPEG2 files. The video service computes again the file extension to manage the trick files and adjust the number of allocated blocks.

Both trick files forward and reverse are generated by the same command. First, the trick file forward is generated by reading the main file. The trick file GOP index is built and kept in memory. During this generation, only the video packets are kept. PCR, PAT and PMT will be regenerated by the MUX in play as for any other streams. The audio packets are discarded. This ensures that there is enough stuffing packets for the PCR reinsertion. For this, a stuffing packet is inserted every 30 milliseconds.

Then using the GOP index, the trick file forward is read GOP by GOP in reverse order to generate the trick file reverse. The same GOPs are present in both files. The only modification done is an update of the video PTS, which must be continuous. Then, the GOP index is written on disk. This avoids reading again the file while generating the second trick file. The GOP index size is: 24 times the GOP number. In the worst case (the file is assumed not to be 1 frame only), there are 2 frames per GOP and 30 frames per second. So for 1 hour in fast forward, the GOP index size is: (24 x 3600 x 30) / 2 = 1296000 bytes. This will be the case for a 4 hour film played at 4 times the normal speed. Therefore, this GOP index can be kept in memory during the trick file generations without risk of memory overflow.

The read and write rates are controlled to conserve bandwidth on the cached disk array. The bandwidth reserved for these generations is a parameter given by the video service. It is a global bandwidth for both read and writes. The number of disk 1/0 per seconds is counted so as not to exceed this bandwidth.

The trick files header update is done once when both the fast forward and fast reverse trick files and the GOP index have been successfully written.

Playing a file is done with the CM_MpegPlayStream class. Fast forward (reverse) can only be requested when we are in the paused state. The current frame on which we are paused is known from the MpegPause class. This frame is located in the GOP index of the trick file. Then the clip start point and length are modified in the Clip instance with the trick file position computed from the beginning of the clip. So, the Clip class handles these trick files in a manner similar to the main file. The current logical block number is updated with the block address in the trick file recomputed from the beginning of the main clip. In fact, a seek is performed in the trick file as it was part of the main file, which is totally transparent for the ClipList and Clip classes. The transition from fast forward to pause is handled in a similar fashion. The clip start and length and the logical block number are again updated. The smooth transitions from pause to fast forward and from fast forward to pause are done in the same way as for regular play. There is a splicing from the pause stream to the play stream.

The class hierarchy for trick file handling is shown in FIG. 28. The MpegFast, MpegFastForward and MpegFastReverse class handles the GOP generation from the initial file. This is the common procedure for building the GOP whatever the source and the destination. RealTimeFastFwd and RealTimeFastRev are the class instantiated when a real time fast forward (reverse) has to be done. They manage the real-time buffer flow to the player. There is a derivation of the methods takeBuffer and returnBuffer which uses the base class to build the GOP in the buffer to be played. The main file access is done using a buffer pool.

TrickFilesGenerate is the class instantiated to generate trick files forward and reverse. It inherits from TrickFileAccess the methods for reading the original file some buffers and for writing the trick file and its meta-data. It inherits from MpegFastForward the methods for building the GOP and for managing the advance in the file.

The computation of the next I frame to play is done by MpegFast, MpegFastForward and RealTimeFastFwd. When a trick file generation command is invoked, a thread is created and started and the generation itself is done off-line. A call-back is sent to the video service when the generation is completed. The class TrickFilesGenerate generates the trick file forward, and then, using the GOP index built in memory, the class TrickFiles Generate generates the trick file reverse.

When there is a transition from play to pause, the only latency issue is related to the buffer queue handled by the player and to the GOP size. The stream can build immediately the active pause GOP, and then this GOP will be sent at the end of the current GOP with a splicing between these two streams.

When there are transitions from pause to regular play or fast forward and fast reverse, a seek in the file is done. This means that the current buffer pool content is invalidated and the buffer pool is filled again. Play can start again while the buffer pool is not completely full, as soon as the first buffer is read. The buffer pool prefilling can continue as a background process. The issue here is that there is a risk to generate an extra load on the cached disk array as well as on the stream server side when the buffer pool is being prefilled.

To avoid too frequent transitions from play to fast forward and fast reverse, there is a limitation of the number of requests per second for each stream. This limitation is part of the management of the video access commands. A minimum delay between two commands is defined as a parameter. If the delay between a request and the previous one is too small, the request is delayed. If a new request is received during this delay, the new request replaces the waiting one. So the last received request is always executed.

The volume parameter (vsparams) file contains these new parameters for the trick mode files:

TrickFileExtensionSize:<percent>:

DefaultFastAcceleration:<acceleration>:

DMtrickFileGen:<mask of reserved DM> (This parameter is a mask of the stream servers that can be chosen to perform the trick file generation. The default value is 0xfffc: all of the stream servers.)

DMtrickFileGenBW:<bandwidth used for trick file generation> (This parameter is the value of the bandwidth effectively used by the stream server for the trick files generation.)

The video service routines are modified to operate upon the EMPG2 files, and in particular to compute the size of the EMPG2 files, to allocate the volume for the main file and the trick files, and to generate the trick files. The volume creation functions (VAPP) and volume access functions (RRP) use the EMPEG2 files in the same way as MPEG2 files. This means that a MPEG2 volume is created on the stream server. Both MPEG2 and EMPEG2 files can be used in the same session or play-list. The session encoding type is MPEG2. In record (or copy-in), the number of blocks allocated for an EMPEG2 file is computed using the percentage of size to add. At the end of record (or copy-in), the number of blocks is adjusted using the number of blocks returned by the stream server (by CMSPROC_GETATTR) and adding the percentage for trick files. The trick files validity and generation date are stored by the video service in the asset structure. The bandwidth allocated to the TrickFilesGenerate command is defined in the volume parameters (vsparams or vssiteparams). The selection of a stream server to generate the trick files takes into account this bandwidth only. If preferred stream servers are specified in vsparams (or vssiteparams), then the selected stream server will be one of these specified stream servers.

In a preferred implementation of the video service software, a new encoding type is created. The encoding type enum becomes:

```
enum encoding-t{
ENC_UNKNOWN    = 0,          /* unknown format */
ENC_RAW        = 1,          /* uninterpreted data */
ENC_MPEG1      = 2,          /* constrained MPEG1 */
EMC_MPEG       = 3,          /* generic MPEG */
ENC_EMPEG2     = 4,          /* MPEG2 with trick files extension */
};
```

- The encoding information accessible by VCMP_EXTENDEDINFO includes information about trick files:

```
struct trickFilesInfo_t{
        ulong_t         generationDate;     /* date/time of the generation of the trick
                files */
        rate_factor_t   acceleration;       /* acceleration factor */
        ulong_t         framesNumber;       /* frames number in each trick file (FWD and
                REV) */
        ulong_t         gopNumber;          /* GOP number of each file */
};

struct EMPEG2info_t{
        MPEG2info_t          MPEG2info;
        trickFilesInfo_t     trickFiles< >;
};

union encodingInfo_t switch (encoding-t enc){
        case ENC_MPEG:
                MPEG2info_t          MPEG2info;
        case ENC_EMPEG2:
                EMPEG2info_t         EMPEG2info;
        default:
                void;
};
```

The video service software includes a new procedure (VCMP_TRICKFILESGEN) for trick file generation, which uses the following structures:

```
struct VCMPtrickgenres_t{
        VCMPstatus_t         status;
        tHandle_t            handle;
};

struct VCMPtrickfilesargs_t{
        name_t          clipname;
        bool_t          overwriteIfExists;
        rate_factor_t   acceleration;
};

VCMPtrickgenres_t          VCMP_TRICKFILESGEN (VCMPtrickfilesargs_t)    =
36,
```

If the trick files already exist and if the boolean overwriteIfExists is true, then the trick files are generated again, in the other case nothing is done. Acceleration is the acceleration as defined and used for the controlled speed play function. It is a percentage of the normal speed, it must be greater than 200 and smaller than 2000. The special value of 0 can be used to generate files with the default acceleration defined in vssiteparams. The procedure starts the generation process. The completion is notified by a callback.

The video service includes a new option to copyin and copyout. The option is added to allow a user to copy all the file or the main asset only. For compatibility with old client programs, the following new procedures are added:

```
VCMPcopyres_t     VCMP_FULL_COPYIN      (copyinargs2_t)      = 37,
VCMPcopyres_t     VCMP_FULL_COPYOUT     (copyoutargs2_t)     = 38,
```

These new procedures take the same interface as the existing one, but are used to copy-in the complete file: meta-data + Asset + trick files.

The video service includes a new procedure VCMP_TRICKFILESGENCOMPLETED, which uses the following structures:

```
struct VCMPtrickfilescomplete_t{
        tHandle_t               handle;
        VCMPstatus_t            status;
};

VCMPstatus_t TRICKFILESGENCOMPLETED (VCMPtrickfilescomplete_t)    = 10,
```

The video service includes new procedures are added for handling trick mode generation arguments, which uses the following structures:

```
struct cms_trick_gen_args {
        Handle_t                Vshandle;
        name_t                  name;
        bool_t                  overwriteIfExists;
        rate_factor_t           acceleration;
        bandwidth_t             reservedBw;
};

cms_status    CMSPROC_GEN_TRICK_FILES (cms_trick_gen_args)            = 34,
struct trick_gen_completed_args {
        Handle_t                Vshandle;
        cms_status              status;
};
void CTLPROC_TRICKGENCOMPLETED (trick_gen_completed_args)        = 8,
```

The video service includes the following option to force the regeneration of trick files even if they exist:

```
nms_content –gentrick <name> [<-f>] [acceleration]
```

Without this option, an error code is returned if the trick files exist. "Acceleration" is an acceleration factor. If it is not present, the default value is taken in vsparams.

The video services include a encoding information access function (nms_content –m). This function produces a displayed output containing, for each trick file generated, the acceleration, the generation date and time, the frames number, and the GOP number.

For the use of an FTP copy function with the trick files, the following new commands are added:

```
nms_content –copyinfull <same arguments as –copyin>
nms_content –copyoutfull <same arguments as –copyout>
```

As described above, transitions between normal play and the fast forward and fast reverse trick modes occur through a pause state. In the pause states, a current frame is repetitively transmitted.

The preferred method of repetitively transmitting a frame is to construct an active pause GOP, and to repetitively transmit the active pause GOP. The active pause GOP contains an I frame and one or more P or B freeze frames. Therefore the I frame is repetitively transmitted, so that the decoder will quickly recover from any loss in synchronization due to a momentary disruption in transmission. The number of frames in the GOP is selected to provide the correct frame rate given the size of the I frame. Stuffing and padding are added to provide precisely the desired bit rate. Therefore, the video buffer verifier (VBV) level at the beginning of the active pause GOP will be the same as the VBV level at the end of the active pause GOP. Consequently, there is no risk of video buffer underflow or overflow when repeating transmission of the active pause GOP for any length of time.

Except in the case of a low bit rate, it is preferred to use "dual motion" P freeze frames in the active pause GOP in order to reduce flicker. If the bit rate is sufficiently low, it may be desirable to use B freeze frames because B freeze frames can be smaller than P freeze frames. To reduce flicker, the P freeze frames are encoded using "dual motion" so that each P freeze frame repeats only one of the two fields in the I frame. By repeating only one field, flicker effects are eliminated from the freeze frames. The I frame, however, will still have a flicker effect, unless the I frame is transcoded to eliminate any difference between the two fields in the I frame. The encoding of the P freeze frames and the transcoding of the I frame to reduce flicker will be further described below with reference to FIGS. 43-48.

With reference to FIG. 29, there is shown a block diagram of the flow of data through the video file server (24 in FIG. 1) to produce a seamless MPEG-2 coded video stream, including active pauses, from a source of MPEG-2 coded video 501. The source 501, for example, is as MPEG-2 file (32 in FIG. 1) in storage of the file server, and the other components in FIG. 29 are hardware and software of the stream server computer (30 in FIG. 1) of the file server. During normal play, the video frames from the source of MPEG-2 coded video are buffered in normal play video buffers 502, and audio data are buffered in audio buffers 504. During a pause, a pause GOP is constructed in pause video buffers 503. A seamless video splicer 505 streams video data from the normal play video buffers 502 or the pause video buffers 503. A transition from the normal play to the pause GOP and a resume from the pause GOP back to normal play is inherently seamless due to the construction of the pause GOP. In this case the seamless video splicer 505 simply functions as a multiplexer. It is also possible, however, to transition from the pause GOP to a new I frame in a seek operation, or to transition from the pause GOP to a fast forward or fast reverse trick mode stream. In these cases the seamless video splicer 505 ensures that the transition will not cause underflow or overflow of the video buffer verifier (VBV).

A transport stream multiplexer 506 combines a video elementary stream from the seamless video splicer 505 with an audio elementary stream from the audio buffers 504 to produce the seamless MPEG-2 coded video stream. The transport stream multiplexer also inserts padding required for the construction of the pause GOP and for seamless splicing. Moreover, the transport stream multiplexer restamps the PTS and PCR values in the audio and video elementary streams with new timestamps from a time base counter 507, and also inserts new continuity values from a continuity counter 508.

FIG. 30 shows the construction of the elementary stream (ES) for the original video (from the source 501 in FIG. 29). The elementary stream includes a sequence header (SH) 510, a GOP header (GH) 511, a picture header (PH) 512, an I frame 513, a B frame 514, and a B frame 515.

FIG. 31 shows the construction of the elementary stream (ES) of a P freeze frame. The elementary stream includes a picture header 521 and a P freeze frame 522. The content of the P freeze frame elementary stream is loaded into payload portions of transport stream packets 523 and 524. The transport steam packets 523 and 524 have respective headers 525, 527 including sync bytes (SB) and respective TS payloads 526, 528.

FIG. 32 shows content of a pause GOP in an elementary stream 531, in a packetized elementary stream (PES) 532, and in a transport stream (TS) 533 before the transport stream multiplexer (506 in FIG. 29), and in a transport stream 534 after the multplexer. The pause GOP ES 531 includes an I frame 535, a first P freeze frame 536, and a second P freeze frame 537. In addition, the pause GOP includes an amount of stuffing 538 selected to make the bit rate of the video transport stream 533 substantially the same but no greater than a desired constant bit rate (CBR) so that the transport stream multiplexer can add padding 538, 539 to obtain the desired constant bit rate.

FIG. 33 is a first sheet of a flowchart showing how an active pause is performed on an I-frame in the MPEG-2 coded video. In a first step 541, the stream server computer (25 in FIG. 1) finds the next I frame in the stream following the time of the pause request. Then in step 542 the stream server computer computes the number of frozen frames needed for the pause GOP from the I frame size and the frozen frame size to obtain a desired frame rate such as 30 frames per second. In particular:

$$\text{FramesPerSecond} = \text{FramesPerPauseGop} * \text{GopsPerSecond}$$

$$\text{GopsPerSecond} = \text{BitsPerSecond} / \text{BitsPerGop}$$

$$\text{BitsPerPauseGop} = \text{PauseGopHeaderBits} + \text{BitsPerIFrame} + (\text{FreezeFramesPerPauseGop} * \text{BitsPerFreezeFrame})$$

These three equations can be solved for the number of frozen frames per GOP. Because the number of frames per pause GOP is one plus the number of freeze frames per pause GOP, the first two of the above three equations give:

$$\text{BitsPerPauseGop} = (1 + \text{FreezeFramesPerPauseGop}) * \text{BitsPerSecond}/\text{FramesPerSecond}$$

The third of the above three equations can be combined with this last equation to eliminate BitsPerPauseGop, and solving for FreezeFramesPerPauseGop results in:

$$\text{FreezeFramesPerPauseGop} = (\text{PauseGopHeaderBits} + \text{BitsPerIFrame} - \text{BitsPerSecond}/\text{FramesPerSecond}) / (\text{BitsPerSecond}/\text{FramesPerSecond} - \text{BitsPerFreezeFrame})$$

In step 543, the stream server computer constructs a pause GOP including the I frame and the frozen frames in the pause video buffers. Then in step 544 the stream server computer performs a seamless splice by switching from the normal play buffer to the pause GOP in the pause video buffers when the I frame is reached. In step 545, the stream server computer inserts padding, new PTS, PCR, and continuity counter values, and selected audio packets into the transport stream to avoid audio and video discontinuities (such as video verifier buffer underflow or overflow, and loss of decoder synchronization) and other artifacts of pausing and splicing, while repeating play of the pause GOP.

In step 546, execution continues to step 547 if the stream server computer receives a "seek" command from the subscriber. In step 547, the stream server computer obtains, from the source of MPEG-2 coded video, a new I frame specified by the seek command. In step 548, the stream server computer computes a new pause GOP including the new I frame, and puts the new pause GOP into the pause video buffers. In step 549, the stream server computer performs a seamless splice from the old pause GOP to the new pause GOP. In step 550, the old pause GOP is deallocated from the pause video buffers. In effect, the old pause GOP is replaced by the new pause GOP. Execution continues from step 550 to step 551 of FIG. 34.

In step 551 of FIG. 34, the normal play video buffer is repopulated beginning with the new I frame. This is done so that the buffer will be filled in advance of the subscriber sending any resume command to resume play beginning at the new I frame. Then in step 552, the stream server checks whether the subscriber has sent a resume command. Step 552 is also reached from step 546 when a seek command is not received from the subscriber. If a resume command has not been received in step 552, then execution loops from step 552 to step 545 in FIG. 33 to play the pause GOP. Otherwise, if a resume command has been received, execution continues from step 552 to step 553. In step 553, the stream server computer performs a seamless splice from the pause GOP to the I frame in the normal play video buffers. If the I frame is an open GOP, each B frame immediately following the I frame is replaced with a frozen B or P frame. For example, it is easy to convert each B frame immediately following the I frame to a B frozen frame. This is done because any B frame immediately following the I frame in an open GOP produces a picture that is different from the picture of the I frame, and that is displayed before the picture of the I frame. Therefore, each B frame immediately following the I frame is replaced with a frozen B or P frame to prevent the picture of the I frame from re-appearing and producing an artifact in which the resumption of play appears to begin at a picture before the picture of the I frame. This will be further described below with reference to FIGS. 40 and 41.

FIG. 35 showing a preferred construction for the normal play video buffers 502 and the pause video buffers 503. Each buffer is arranged as a carousel of buffers, in a circular arrangement. Each buffer, such as the buffer 561, provides, for example, 64 kilobytes of memory. Content of some of the buffers is shown to illustrate a switch from play to pause and back. The carousel of pause video buffers can be instantiated upon receipt of the pause command.

In a preferred method, if the current state is playing, the next I frame is determined using a GOP counter and the current position in the MPEG-2 source file. The GOP index in the file provides the I frame number, its size and its offset. Then the PTS (and the DTS) of the paused frame is computed using the first PTS value stored in the meta-data, of the I frame when the pause started, and the frame number. The position offset of the GOP inside the circular data buffer is determined using the packet number in the original file and the number of complete data buffers up to the one that contains the GOP header. The preparation of the pause GOP is done immediately by instantiating (or resetting if it already exists) an MpegPause object. Usually some time must pass before switching the streaming from the normal play buffers to the pause video buffers. In particular, the normal play buffers are used until the next GOP header is reached (i.e. the GOP starting on the same I frame as the one used to build the frozen GOP). When this point is reached, all the preceding packets of the current buffer are copied in an ancillary buffer. Following audio packets are also copied until the PTS of the audio frame is greater than the PTS of the paused frame, as further described below with reference to FIG. 52. The ancillary buffer is returned and the seamless video splicer switches over to the pause buffer source. The current buffer is left in the normal play buffer. Its field "dataOffset" is adjusted to the resume position. Thus, the normal play buffers are ready for a resumption of play restarting from the paused I frame.

To begin sending video data from the paused GOP, the MpegPause object needs the following data: a pointer to an MpegFrozenFrame object, a pointer to an MpegFSIO object associated to the clip, the pause frame offset, its frame number, its size, its PTS, the frame rate, and the stream bitrate. At its instantiation, the MpegPause object builds the pause GOP. It contains the I frame and some frozen P frames.

The number of frozen frames to add is implied by buffer considerations: the buffer level must be the same at the beginning of each GOP and equal to the original one.

This means that after each GOP, the same time should had been added to PCR and PTS. The number of frozen frames is calculated from the frozen frame size and the I frame size. Unfortunately, it is unlikely that the result matches an integer. Thus, the GOP is completed with stuffing. The stuffing, however, has a precision of one packet which is not accurate enough. That is why additional padding is added by the MUX in order to compensate for the variations in the bit rate. The creation of the pause GOP can be done without consideration of the new PCR and PTS values and without consideration of the VBV buffer level. The pause GOP is created with a PCR place holder such as a zero value. Each buffer sent to the transport stream multiplexer is processed to add selected padding, to restamp the PTS and the DTS of each video frame, and to restamp the continuity counts.

Due to the buffer processing made at pause time, the normal play video buffers are ready to restart in response to a resume command, without any additional processing. Especially, it is not necessary to repopulate it, which avoids additional I/O and reduces response time. If the clip has been encoded using open GOP; the first GOP to send during a resume is not exactly the pause GOP. In this case it has to contain the I frame, the following B frames and possibly some frozen P frames. Any B frame in an open GOP following the I frame and pointing backwards to a P frame, is replaced by B freeze frame. This can be done during the pause, to avoid any delay in responding to a resume command. The switching from the pause video buffer to the normal play buffer is driven by the MpegPause object so that it occurs precisely at the end of the pause GOP. The seamless video splicer ensures a smooth resume.

FIG. 36 shows a graph of the level of a video buffer verifier as a function of time during an MPEG-2 coded video stream including normal play, a first pause, a seek, a second pause, and a resumption of the normal play. This graph shows that the level at the beginning of a pause is substantially the same as the level at the end of the pause. This is a result of the fact that the pause consists of the frames of an integral number of pause GOPs, and each pause GOP is constructed so that the level at the beginning of the pause GOP is the same as the level at the end of the pause GOP.

FIG. 37 shows a sequence of video frames in an original stream of MPEG-2 coded data. The frames are shown in transmission order. Each sequence of the form "IBBPBB" is a GOP. Each GOP of this form can be open or closed depending on whether or not the B frame immediately following the I frame references the last frame of the preceding GOP. For example, If the GOP is closed, then the I frame will be presented first, and if the GOP is open, then the B frame following the I frame will be presented first.

FIG. 38 shows how the original stream of FIG. 37 is modified during an active pause upon a closed GOP for a first case of a play followed by a pause. A first pause occurs on the I frame I1 of the GOP "I1 B2 B3 P4 B5 B6." The first pause GOP has the I frame I1 immediately followed by two P freeze frames. During the first pause, the first pause GOP is played twice. Playing is resumed on the GOP "I1 B2 B3 P4 B5 B6." Then a second pause occurs on the I frame I7. The second pause GOP has the I frame I7 immediately followed by two P freeze frames. During the second pause, the second pause GOP is played twice. Playing is resumed on the GOP "I7 B8 B9 P10 B11 B12."

FIG. 39 shows how the original stream of FIG. 37 is modified for an active pause upon a closed GOP for a second case of a play followed by a pause, followed by a seek. In this case, the seek is a direct transition in play of a second pause GOP having the I frame I7 and two immediately following P freeze frames, to play of a third pause GOP having the I frame I19 and two immediately following P freeze frames.

FIG. 40 shows how the original stream of FIG. 37 is modified for an active pause upon an open GOP for a first case of a play followed by a pause. In this case it is assumed that the GOP "I7 B8 B9 P10 B11 B12" is an open GOP so that the presentation order of the frames in the GOP would be B8 B9 I7 B11 B12 P10. Upon resuming from the second pause, the B frame B8 is replaced with a B freeze frame 511, and the B frame B9 is replaced with a B freeze frame 512. If the B frames B8 and B9 were not replaced with freeze frames, then the sequence of frames presented upon resuming from the second pause would be I7 P-freeze P-freeze B8 B9 I7 P10. Because the pictures of the frames B8 and B9 would appear to be different from the picture of the frame I7, the pause of the frame I7 would appear to be interrupted by the frames B8 and B9. In fact, the pictures of the frames B8 and B9 would not be what these pictures were intended to be, because the frame B8 should forward reference the picture P4, and not a P frozen frame reproducing the picture of I7. This artifact is avoided by replacing the frames B8 and B9 with the B freeze frames 511 and 512.

FIG. 41 shows how the original stream of FIG. 37 is modified for an active pause upon an open GOP for a second case of a play followed by a pause, followed by a seek. In this case, the GOP "I19 B20 B21 P22 ..." is assumed to be an open GOP so that the frames B20 and B21 would precede I19 in presentation order. Upon resuming from the third pause on I19, the frames B20 and B21 are replaced with B freeze frames 513 and 514.

FIG. 42 shows a class structure for various program objects for implementing the configuration of FIG. 29 in the video file server of FIG. 1. A BufferSource object dynamically allocates the buffers of memory in the stream server computer. The class MpegPause is derived from BufferSource to handle active pause. The class MpegPause is instantiated either by a CM_MpegPlayStream or by a CM_MpegPlayListStream depending on whether the MPEG stream is for playing from a list of clips. The MpegPause object is instantiated the first time a pause is required. Each time the pause position changes, the MpegPause object is reset. Then this MpegPause object is used by the player as its buffer source, until a resume occurs. Also shown in FIG. 42 is a class MpegFrozenFrame that is instantiated to produce a frozen frame, a class MpegSplicer that is instantiated to produce a seamless splice, and classes MpegFast, MpegFastForward, and MpegFastReverse for the trick modes.

As described above, the pause GOP is constructed of an I frame followed by one or more freeze frames. As in the current TV systems, one frame is made of two interlaced field pictures that do not have the same time instance, so I-frame picture contains information from two instants. This results in the famous flickering effect if the picture is simply repeated during the pause period. To eliminate this undesired effect, the two fields of each freeze frame should be predicted from a same field of the I-frame (top or bottom). The advantage of such a solution is its simplicity. The freeze frames may be generated in advance or in real time with only some hundred of byte per frame while I-frame needs no modification. However, this solution does not entirely eliminate the flicker effect because each time the GOP is repeated, the flickering effect will appear briefly for the I-frame in the GOP.

FIG. 43 shows a preferred way of implementing a P-freeze frame in order to freeze a single field of an I-frame. The I frame 521 includes a top field 522 at a time $T=T_0$ and a bottom field 523 at a later time $T=T_0+1/2FR$. If the picture encoded by the I frame 521 includes objects in motion, then the top and bottom fields 522 and 523 will appear substantially different, and cause the flicker effect when the picture of the I frame is repeated during the pause. The flicker effect can be reduced by encoding the freeze frames as a dual-motion P freeze frame 524 in which the top field 525 is predicted with no motion from the top field 522 of the I frame 521, and in which the bottom field 526 is predicted with motion from the top field 522 of the I frame.

If the flicker effect of the freeze frames is eliminated by using dual-motion encoded P freeze frames, it may be desirable to lengthen the pause GOP structure to include more P freeze frames to suppress the flicker effect remaining in the I frame. This has the disadvantage of exposing the decoder to a longer recovery time if the decoder would happen to lose synchronization due to an error in transmission from the video file server to the decoder. An alternative solution is to replace the I frame in the pause GOP with a transcoded I frame in which the two fields are substantially similar to each other. For example, each of the two fields in the transcoded I frame is the same as one of the two fields in the original I frame. A specific example is shown in FIG. 44. A transcoded I frame 527, designated I', is produced including the top field 528 that is the same as the top field 522 of the I frame 521, and a bottom field 529 that is the same as the top field 522 of the I frame 521. Because the both the I frame 521 from the original MPEG-2 coded video and the transcoded I frame 527 are compressed, the production of the transcoded I frame 527 from the original I frame 521 can be relatively difficult or relatively easy depending on how the original I frame 521 happens to be encoded.

FIG. 45 shows how the transcoding of FIG. 44 is performed for either a frame coded I frame or a field coded I frame, and for field DCT or for frame DCT encoding for a picture coded I frame. In a first step 531, execution branches depending on the picture coding type, specified by a "picture_structure" parameter in the original coded I frame. The picture can be coded either as two field-pictures (picture_structure='01' and '10') or as one field picture as one frame-picture (picture_structure='11'). Execution branches to step 532 if the original I frame is not frame-picture coded. In this case, each field of the original I frame is coded separately. Therefore, in step 532, the encoding for the first field of the transcoded I frame is produced by copying the encoding of the first field from the original I frame, and the encoding for the second field of the transcoded I frame is produced as a fully-predicted P field of the first field from the original I frame. Alternatively, the encoding for the second field of the transcoded I frame could be produced by copying all coded slices from the first coded field-picture without changing picture coding type. By encoding the second field as a predicted P field, however, the size of the transcoded I frame can be reduced, which is better from the point of view of the buffer and bandwidth management

In step 531, execution continues to step 533 if the original I frame is frame-picture coded. In this case, the luminance blocks of each macroblock of the frame are DCT encoded either on a frame basis or a field basis. In step 533, a macroblock counter is set to zero. In step 534, the next macroblock is extracted from the original I frame. In step 535, execution branches depending on whether the luminance blocks in the macroblock are frame DCT encoded or field DCT encoded. This is specified by a "DCT_type" parameter in the encoded I frame. If "DCT_type" is 1, then the luminance blocks of the macroblock are field DCT encoded and not frame DCT encoded, and execution branches to step 536. In step 536, the first field luminance blocks for the transcoded I frame are the same as the first field luminance blocks for the original I frame, and the second field luminance blocks for the transcoded I frame are the same as the first field luinance blocks for the original I frame. Therefore, to convert the original encoded I frame to the transcoded I frame, no modification is needed for the first field luminance blocks, and the second field luminance blocks are replaced by coded first field luminance blocks. Although the DC coefficient of the DCT block does not change, it will need to be reencoded if the DC coefficient of the previous DCT block is changed because each DC coefficient is encoded as a prediction from the DC coefficient of the previous macroblock. The variable-length coded bits of the AC coefficients may be directly copied when copying the block from the original I frame to the transcoded I frame.

In step 535, if the luminance blocks in the macroblock are frame DCT encoded, then execution continues to step 537. In step 537, the luminance blocks in the macroblock from the original I frame must undergo a relatively complex process of field line replacement in order to produce the luminance blocks for the transcoded I frame from the original I frame. This process of field line replacement will be further described below with reference to FIGS. 46 to 48.

In a typical case of 4:2:0 format, the chrominance blocks Cb and Cr are encoded on a frame basis regardless of whether the luminance blocks for the macroblock are frame DCT encoded or field DCT encoded. Therefore, as indicated in step 538, the chrominance blocks in the transcoded I frame are the same as the chrominance blocks in the original I frame, so that no modification of the chrominance blocks is need for production of the transcoded I frame.

After steps 535, 536 or 537, execution continues to step 539. In step 539 the macroblock counter is incremented. Then in step 540, execution loops back to step 534 if the end of the frame is reached. Otherwise, the transcoding procedure is finished.

FIG. 46 shows the use of field line replacement for the case of a pause GOP including an I frame 551 that is picture coded with frame DCT. A transcoded I frame 554, designated I', is produced including the top field 555 that is the same as the top field 552 of the I frame 551, and a bottom field 556 that is the same as the top field 552 of the I frame 551. The DCT of the frame coded 8x8 pixel blocks of both the I frame 551 from the original MPEG-2 coded video and the transcoded I frame 554 intermingle pixel blocks for the top field with pixel blocks for the bottom field. Therefore, a relatively complex process of field line replacement 557 is needed for producing the transcoded I frame 554 from the original I frame 551.

The field line replacement of FIG. 46 could be performed by decoding and re-encoding the pixels of the I frame. Let P(r,c) be the original luminance component of the pixel at the position with row_number=r and column_number=c, then the luminance P'(r, c) of the new image is calculated with:

If progressive_sequence=0:

$P'(2*r, c) = P'(2*r+1, c) = P(2*r+1\text{-}top\_field\_first, c)$

If progressive_sequence=1:

$P'(2*r, c) = P'(2*r+1, c) = P(2*r, c)$

In the original I frame and the transcoded I frame, however, the luminance components are DCT encoded. Therefore, if the field line replacement were to be done by replacing the luminance components of either the even or odd lines of pixels in the frame, it would be necessary to decode the original I frame DCT coefficients into the pixel domain, then replace one field by another, and then reencode the pixels into the DCT domain to produce the transcoded I frame. Such a direct approach would require an excessive number of computations since a two-dimensional (2D) inverse Discrete Cosine Transform (IDCT) would be needed for converting DCT coefficients to pixels in the first step and the reverse operation (2D-DCT) would be needed in the last step. 2D-IDCT and DCT are complex operations (for example 176 multiplications plus some hundreds of additions). However, some simplifications can be achieved in the compressed I-frame conversion.

Let B(r, c) be the element of the considered block of 8x8 pixels and B'(r, c) be its homologue in the new picture. To be more general, we ca write the following equations:

$B_d'(2*i, j) = B'(2*i + 1, j) = B'(2*i+d, j)$ for $4 > i \geq 0$ and $8 > j \geq 0$

With $d = (1 - top\_field\_first) * (1 - progressive\_sequence) \Rightarrow \quad d = 0 \text{ or } 1$

It is well known that the 2D-DCT and IDCT are separable transformations, i.e., a 2D transformation may be realized by two cascaded 1D operations. For example, for calculating a 2D-DCT (or IDCT) transform on a block of 8x8 pixels, one can make 1D-DCT (or IDCT) transform on each row (or column) and then make 1D-DCT (or IDCT) transform on each column (or row) of the resulting block. So the above problem is simplified to a one-dimension problem as follows:

If

$g_d(2x) = g_d(2x+1) = f(2x+d)$

and $F(u) = DCT1D\,[f(x)]$ with all $F(u)$ known,

then how may one generate $G_d(u)=DCT1D\,[g_d(x)]$?

FIG. 47 shows how the field line replacement is performed using a one-dimensional (1D) IDCT 561, followed by a field line replacement 562, followed by a one-dimensional (1D) DCT 563. So three operations are necessary.

Now, consider the definitions of 1D-DCT and 1D-IDCT:

1D-DCT:

$$F(u)=\frac{1}{2}C(u)\sum_{x=0}^{7} f(x)*\cos\left(\frac{(2x+1)u\pi}{16}\right)$$

ID_IDCT:

$$f(x)=\frac{1}{2}\sum_{u=0}^{7} C(u)*F(u)*\cos\left(\frac{(2x+1)u\pi}{16}\right)$$

$$\text{where}\quad C(u)=\begin{cases}\dfrac{1}{\sqrt{2}}, & \text{if}\quad u=0\\ 1, & \text{otherwise}\end{cases}$$

Then one can write:

$$f(2x+d)=\frac{1}{2}\sum_{u=0}^{7} C(u)*F(u)*\cos\left(\frac{(4x+2d+1)u\pi}{16}\right)$$

and

$$G_d(u)=\frac{1}{2}C(u)\sum_{x=0}^{3}\left\{g_d(2x)*\cos\left(\frac{(4x+1)u\pi}{16}\right)+g_d(2x+1)*\cos\left(\frac{(4x+3)u\pi}{16}\right)\right\}$$

$$G_d(u)=\frac{1}{2}C(u)\sum_{x=0}^{3}g_d(2x)*\left\{\cos\left(\frac{(4x+1)u\pi}{16}\right)+\cos\left(\frac{(4x+3)u\pi}{16}\right)\right\}$$

Note: $\cos(\alpha)+\cos(\beta)=2*\cos\left(\frac{\alpha+\beta}{2}\right)*\cos\left(\frac{\alpha-\beta}{2}\right)$

$$G_d(u)=C(u)\sum_{x=0}^{3}f(2x+d)*\cos\left(\frac{(2x+1)u\pi}{8}\right)*\cos\left(\frac{u\pi}{16}\right)$$

$$G_d(u)=\frac{1}{2}C(u)*\cos\left(\frac{u\pi}{16}\right)*\sum_{x=0}^{3}\sum_{v=0}^{7}C(v)*F(v)*\cos\left(\frac{(4x+2d+1)v\pi}{16}\right)*\cos\left(\frac{(2x+1)u\pi}{8}\right)$$

$$G_d(u)=\frac{1}{2}C(u)*\cos\left(\frac{u\pi}{16}\right)*\sum_{v=0}^{7}C(v)*F(v)*\left[\sum_{x=0}^{3}\cos\left(\frac{(4x+2d+1)v\pi}{16}\right)*\cos\left(\frac{(2x+1)u\pi}{8}\right)\right]$$

Then, we can give the following equations:

$$G_d(u)=\sum_{v=0}^{7}K_d(u,v)*F(v)$$

with $K_d(u,v)=\frac{1}{2}C(v)*C(u)*\cos\left(\frac{u\pi}{16}\right)*\sum_{x=0}^{3}\cos\left(\frac{(4x+2d+1)v\pi}{16}\right)*\cos\left(\frac{(2x+1)u\pi}{8}\right)$

One can note that $K_d(u, v)$ is a constant for each given couple (u,v) and a fixed d. As a consequence, a simple 1D-transformation defined above is sufficient to convert *F(u)* to *G(u)*. This new solution is show in FIG. 48. This is a linear transformation represented as a matrix multiplication 565 that converts the encoded DCT coefficients of the I frame to the encoded coefficients of the transcoded I frame having the line replacement.

The two matrices $K_d(u, v)$ with d=0 and 1 have interesting properties which may be exploited to simplify the above transformation.

The first property is that the two matrices are highly correlated. Their relationship can be expressed by the following equation:

$$K_{1-d}(u,v) = (-1)^{(u+v)} * K_d(u,v), \qquad \forall u, v$$

The second property is that any row except the first one can be calculated with its symmetrical one (with regard to the fourth row):

$$K_d(8-u, v) = -tan(u*\pi/16) * K_d(u, v) \qquad \textit{for } u=1,2,3,4,5,6,7 \textit{ and } \forall\, d, v$$

As a consequence,

$$G_d(8-u) = -tan(u*\pi/16) * G_d(u) \qquad \textit{for } u=1,2,3,4,5,6,7 \textit{ and } \forall\, d$$

<u>The third property is that the fourth row is always a zero row:</u>

$$K_d(4, v) \equiv 0, \qquad \forall v$$

and

$$G(4) \equiv 0$$

The element values of the matrices K may be easily calculated directly with its definition formula or with a further developed one. The values of the matrices K with d=0 and 1 are respectively given in Table 1 and Table 2 below.

Table 1. Values of $K_d(u, v)$ with d=0

u\v	0	1	2	3	4	5	6	7
0	1	0.18024	0	0.21261	0	0.31819	0	0.90613
1	0	0.96194	0.34676	0	0.37533	0	0.83715	-0.19134
2	0	-0.06897	0.85355	0.47421	0	0.70970	-0.35355	-0.34676
3	0	0	-0.12177	0.69134	0.76818	-0.46194	-0.29397	0
4	0	0	0	0	0	0	0	0
5	0	0	0.08136	-0.46194	-0.51328	0.30866	0.19642	0
6	0	0.02857	-0.35355	-0.19642	0	-0.29397	0.14645	0.14363
7	0	-0.19134	-0.06897	0	-0.07466	0	-0.16652	0.03806

Table 2. Values of $K_d(u, v)$ with d=1

u\v	0	1	2	3	4	5	6	7
0	1	-0.18024	0	-0.21261	0	-0.31819	0	-0.90613
1	0	0.96194	-0.34676	0	-0.37533	0	-0.83715	-0.19134
2	0	0.06897	0.85355	-0.47421	0	-0.70970	-0.35355	0.34676
3	0	0	0.12177	0.69134	-0.76818	-0.46194	0.29397	0
4	0	0	0	0	0	0	0	0
5	0	0	-0.08136	-0.46194	0.51328	0.30866	-0.19642	0
6	0	-0.02857	-0.35355	0.19642	0	0.29397	0.14645	-0.14363
7	0	-0.19134	0.06897	0	0.07466	0	0.16652	0.03806

With the above tables, we can easily verify the properties of the matrices K_d. Moreover, the most useful inter-rows relations are as follows:

$$G_d(5) = -0.66818 * G_d(3)$$
$$G_d(6) = -0.41421 * G_d(2)$$
$$G_d(7) = -0.19891 * G_d(1)$$

In each $K_d(u, v)$ table, we can find 27 zero values and one value that equals 1. This means, to compute one column, at most 36 multiplications will be needed, i.e., for one complete block, at most 36*8=288 multiplications are needed. Moreover, considering the equations above for the correlation between paired lines, only the non-zero coefficients in the five first rows are necessary. They count only 20 (the value 1 excluded). Taking into account the three multiplications in the last three equations above for the correlation between paired lines, the maximal total number of multiplications becomes:

*23*8 = 184 multiplications / block.*

The number of additions will be:

*(20-4)*8 = 128 Additions / block.*

In the real-world applications, the average numbers of multiplications and additions will be significantly reduced because there will be many zero AC coefficients in the DCT blocks. Moreover, the multiplications may be further substituted with tabulation. For example, if a decoded DCT coefficient $F(v)$ has a row number equal to 2 we can read from a pre-computed table the following values:

$$\text{Table\_2}[F(v)] \Leftrightarrow \{\, F(v)*0.34676,\ F(v)*0.85355,\ F(v)*0.12177 \,\}$$

$K_d(u, v)$ may refer to the same tables. If we consider only non-zero values contained in the first five rows (with the value 1 excluded) and those in the last three equations for the correlation of paired lines, the total size of the tables will be:

23 * DCT_coefficient_dynamical_range * size_of_individual_value

In MPEG-2 video, intra ACs range from –1024 to 1023 and size_of_individual_value may be equal to 4 bytes. Therefore,

$\Rightarrow$ *tables_size = 23 * 2048 * 4 = 188,416 bytes*

FIG. 49 is a flow diagram for the field line replacement in the DCT domain using the matrix multiplication of FIG. 48. The coded bit stream enters the variable-length decoding function 571 whose outputs are run and level. The run is used in a function 572 that computes the coefficient index in the DCT block in accordance with the coefficient scanning order, and the resulting coefficient index is used in an indexing function 573 to obtain the row and column positions of the coefficient in the 8x8 block of coefficients for the DCT block.

The level is used in an inverse quantization function 574 to produce a dequantized level that is used, together with the row number, to index a multiplicaton table 575 to obtain the multiplication results K(u,v)*F(v) for all non-null K(u,v)'s in the first five rows in the Table 1 above. The multiplication values are then added in an adder 576 to produce a sum for the corresponding column of the DCT block.

When the VLD encounters the end_of_block code, the results from rows[1:3] in temporary memory 577 are selectively complemented by a function 578 to produce the coefficients for the rows numbered 5, 6 and 7 in accordance with the $G_d(8\text{-}u) = -\tan(u*\pi/16) * G_d(u)$ or with the equations G(5)= -0.66818*G(3), G(6)= -0.41421*G(2) and G(7)= -0.19891*G(1). Alternatively, the coefficients for the rows numbered 5, 6 and 7 could be calculated by multiplication or multiplication table. The 4th entire row in the DCT block is always a zero-row. A final function 579 encodes the resulting block of DCT coefficient will then be encoded into coded bit stream with the usual run, level and variable-length coding.

The overall processing complexity is a function of the number of non-zero DCT coefficients per block. Therefore, the processing time will be different for the highly compressed I-frames and for those with low compression ratios. However, the transcoding of the I frame in real time is possible in most cases. When it is not possible to transcode the entire I frame before playing the pause GOP, it is possible to make a progressive substitution within a very short transition time. For example, at the beginning of the pause, *I'* will be simply a copy of *I*. Each time that there is some available CPU processing time in the stream server computer before sending a new paused frame, the additional time is used to transcode one or more additional slices of the I frame of the pause GOP until all slices have been processed.

When progressive substitution is used, there would be a gradual reduction in the flicker effect. If the flicker effect persists for a short but noticeable amount of time, then a two-step substitution could be used. In this case, as shown in FIG. 50, the first I-frame (I_0) 581 in the pause is a partially transcoded version of the pause I frame in the original MPEG-2 video. Dual-motion encoded P freeze frames 583, 584 are inserted after the first play of the pause GOP 582 until the entire pause I frame has been transcoded to produce frame I' in the pause GOP. Then the pause GOP is replayed, producing the sequences 585 and 586 during the pause. The first I-frame (I_0) 581 in the pause will introduce a partial flickering effect and the remaining pictures will provide a perfectly stationary image. In case of transmission error, however, resynchronization of the decoder is only ensured after transcoding of the entire frame.

FIG. 51 shows a flowchart of programming for the progressive transcoding and the two-step transcoding methods for reducing flicker. In a first step 601, the I frame selected for pausing is placed in the pause video buffers. Then, in step 602, a background task is activated that replaces a field in the selected I frame by transcoding on a slice-by-slice basis. For example, the background task begins with the first slice of the I frame, transcodes the first slice to remove the encoded field 1 pixels and replace them with encoded field 0 pixels, replaces the original slice in the I frame with the transcoded slice, and then repeats the transcoding and replacement operations successively for each of the remaining slices in the I frame until the entire I frame has been transcoded. In step 603, the first play-out of the pause GOP from the video buffer occurs. The I frame of this pause GOP as played out most likely will have a number of slices that have been transcoded, and possibly some slices that have not been transcoded. In step 604 execution branches depending on whether or not further transcoding of slices in the I frame has been preselected to occur according to either the progressive method or the two-step method. For example, the branch condition for step 604 could be specified by an attribute bit of the MPEG clip or file that is being played. For the progressive method, execution continues from step 604 in the normal fashion, by replaying the pause GOP, while the background task transcodes the remaining slices. For the two-step method, execution continues to step 605. In step 605, if the background task has finished transcoding all of the slices in the pause GOP, then execution continues in the normal fashion to replay the pause GOP. Otherwise, if the background task has not yet finished transcoding all of the slices in the pause GOP, execution continues from step 605 to step 606. In step 606, a P freeze frame is played, and execution loops back to step 605. In this fashion, P freeze frames are played until the entire I frame of the pause GOP has been transcoded.

FIG. 52 shows a preferred alignment of audio presentation units (APUs) with video presentation units (VPUs) during a pause. During a pause, there is a quiet interval 591 during which no audio presentation units are sent. The last audio presentation unit before the quiet interval 591 is designated APU_y and it is the last audio presentation unit to begin during the video presentation unit (VPU-Ix) of the I frame (Ix) that begins the pause and that is the I frame of the pause GOP. During a resume on the I frame (Ix) of the pause, the first audio presentation unit (APU_{y-1}) to begin after the quiet interval 591 is the first audio presentation unit to end during the video presentation unit (VPU-Ix) of the I frame (Ix) of the pause. This alignment and selection of audio presentation units tends to provide the most natural interruption of the audio stream during the pause.

As described above, it is desirable to repetitively transmit an I frame during a pause so that the decoder will quickly recover from any loss in synchronization due to a momentary disruption in transmission. In this case, it is natural for the pause to begin on a selected I frame. This means, however, that in the general cases a pause will occur at most about 0.5 seconds (on the order of a GOP duration) after receiving the pause command. This maximal latency could be largely reduced if pausing is also allowed on P frames. This would be desirable, for example, if the MPEG stream were to be transmitted over a channel having a very low transmission error rate or if the MPEG stream were to be decoded with a special decoder that would be highly tolerant of a loss of synchronization.

In order to pause on a P frame, an "infinite" GOP is constructed. This means that only one GOP is sent during all time the stream remains paused. Consider, for example, the following MPEG video sequence in the decoding order:

I2 B0 B1 P5 B3 B4 P8 B6 B7 P11 B9 B10 I14 B12 B13 P17 B15 B16 ...

where each number immediately following the letter I or P or B corresponds to the display order. When a pause is intended on 5th image (B4), the following sequence is sent:

I2 B0 B1 P5 B3 B4 *p2 b0 b1 p5 b3 b4 .... p(n-1) b(n-3) b(n-2)* P8 B6 B7 P11 B9 B10 I14 B12 B13 P17 B15 B16 ...

where *p* and *b* respectively represent P-frame and B-frame and *p2 b0 b1 p5 b3 b4 .... p(n-1) b(n-3) b(n-2)* represent frozen images which repeat the previous reference picture (I or P frame). In this example, the decoded frozen images are the same as the decoded P5. The number *n* corresponds to the pause duration in number of frames. Pause delay is 1 frame period (instead of 10) Each frozen picture will have the same number of bytes as those entering into the decoding buffer during one frame period:

bitrate / frame_rate / 8

In short, pausing on a P frame reduces response time and enhances the accuracy of the pause command at the risk of loss of decoder synchronization if any transmission error occurs. The decoder will be able to resynchronize after a resume. Because the desirability of this feature depends on an environment where the likelihood of a transmission error is very low, the feature is best implemented as an option. For example, the feature could be specified by a flag for MPEG processing options, and this flag could be validated depending on the server environment.

In view of the above, there has been described a pause and resume function that that delivers a valid MPEG data stream without video buffer underflow or overflow. In the preferred implementation, the transition from play to pause is seamless, the transition from pause to play is seamless, the result of the paused state is a perfectly stilled picture, and the result does not depend on decoder performance. The stream is compliant with the

MPEG-2 standard. There are no PCR or PTS discontinuities. The initial bit rate as encoded is preserved. There are no discontinuities in the TS packets. There are neither any audio discontinuities nor decoder de-synchronization. Consequently, the MPEG data stream is paused and resumed without the introduction of objectionable artifacts.